МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ

по учебной практике

Тема: Генетические алгоритмы

Кузнецова С.Е.
Максимова Е.Д.
Студентки гр. 3341 _____ Чинаева М.Р.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург
2025

Цель работы.

Создание финального по визуалу GUI, создание рабочей версии генетического алгоритма

Задание №18.

Задача о клике.

В заданном графе G = (V,E) необходимо найти клику максимального размера.

Распределение ролей в бригаде.

Максимова Екатерина: Реализация GUI

Чинаева Маргарита: Реализация основного алгоритма и подготовка для связи с GUI

Кузнецова Светлана: Загрузка и выгрузка данных и вспомогательные функции для основного алгоритма (проверка смежности, прохождение по соседям, проверка подграфа, является ли он кликой и т.п.) + генерация случайных данных

Описание текущей реализации генетического алгоритма:

Класс *GeneticAlgorithm* - реализует генетический алгоритм для поиска максимальной клики в графе. Использует эволюционные методы для нахождения оптимального решения задачи о максимальной клике.

Поля класса:

Graph graph - Граф, в котором ищется максимальная клика

Parameters params - Параметры работы генетического алгоритма

float current_mutation_prob_chrom - Текущая вероятность мутации хромосомы

float current_mutation_prob_gene - Текущая вероятность мутации отдельного гена

int current_crossover_points - Текущее количество точек кроссовера

int generation - Номер текущего поколения

int stagnation_count - Счетчик поколений без улучшения лучшего решения

int best_fitness - Найденный максимальный размер клики

list[int] best_chromosome - Лучшая найденная хромосома

int n - Количество вершин в графе

int max_degree_original - Максимальная степень вершины в исходном графе

Основные методы:

__init__(self, graph: Graph, params: Parameters) – Инициализирует алгоритм с заданным графом и параметрами, выполняет преобразование графа и генерацию начальной популяции.

generate_chromosome(self) -> List[int] – Генерирует случайную хромосому, представляющую клику, использует жадный алгоритм с случайным выбором вершин среди вершин одной степени.

generate_initial_population(self) -> List[Individual] – Создает начальную популяцию заданного размера.

select_parents(self) -> List[Individual] – Выбирает родителей для скрещивания методом рулетки с масштабированием приспособленности.

crossover(parent1, parent2) -> Tuple[List[int], List[int]] - Выполняет кроссовер с несколькими точками разрыва, попеременно копируя сегменты хромосом родителей.

mutate_and_repair(chromosome) -> List[int] – Применяет мутации и восстанавливает хромосому до валидной клики.

select_new_population(current_pop, offspring) -> List[Individual] – Формирует новую популяцию с сохранением разнообразия, выбирая особи, максимально отличающиеся от уже выбранных.

_update_best_solution() – Обновляет лучшее решение при обнаружении улучшения.

_reduce_parameters() – Уменьшает параметры алгоритма на заданный процент от начальных значений.

should_stop() -> bool – Проверяет условия остановки алгоритма.

next_generation() – Выполняет одну полную итерацию генетического алгоритма:

1. Выбор родителей
2. Генерация потомков
3. Формирование новой популяции
4. Обновление лучшего решения и списка поколений
5. Уменьшение параметров (при необходимости)

get_best_solution() -> Tuple[int, List[int]] – Возвращает лучшее решение в исходной нумерации вершин.

_hamming_distance() – Вычисляет нормализованное расстояние Хэмминга

Обработка и хранение данных, вспомогательные функции

Были реализованы классы для хранения данных, использующихся в работе алгоритма, и вспомогательные функции.

1. Класс *Graph* – отвечает за работу с графом, его загрузкой, имеет вспомогательные функции для работы с графом

Поля класса:

list[set[int]] adj_list – начальный граф в виде списка смежности (списка множеств)

int n – его размер (количество вершин)

list[set[int]] transformed_adj – преобразованный граф в виде списка смежности – граф с переназначенными вершинами, отсортированными по убыванию степеней в исходном графе

list old_to_new – список для преобразования старых индексов (вершин графа) в новые

list new_to_old – список для преобразования новых индексов (вершин графа) в старые

Основные методы:

from_adj_matrix(matrix: list[list[int]]) -> 'Graph' – строит граф в виде списка смежности из матрицы смежности

load_from_file(file_path: str) -> 'Graph' – загружает граф через json-файл, в котором граф представлен в виде "вершина: список соседей"

random_graph(n: int, p: float) -> 'Graph' – генерирует случайный список смежности размера n с вероятностью p для каждого ребра i – j и строит по нему граф

transform_by_degree(self) – преобразует граф, переупорядочивая вершины по убыванию степени. Создает списки для отображения старых индексов в новые, новых в старые

transform_to_original(self, sorted_chromosome: list[int]) -> list[int] – преобразует хромосому из преобразованной нумерации в исходную нумерацию вершин графа

transform_to_sorted(self, original_chromosome: list[int]) -> list[int] – преобразует хромосому из исходной нумерации в преобразованную

get_subgraph(self, chromosome: list[int]) -> list[set[int]] – строит подграф на основе хромосомы (для исходного графа). В дальнейшем будет необходима для построения подграфа в GUI.

all_neighbors(self, v: int) -> list[int] – возвращает список соседей вершины v в преобразованном графе

has_edge(self, u: int, v: int) -> bool – проверяет наличие ребра между u и v в преобразованном графе

degree_in_subgraph(self, included: list[int]) – вычисляет степени вершин в подграфе (в преобразованном графе)

is_clique(self, chromosome: list[int]) -> bool – проверяет, задают ли включенные в хромосому вершины клику в преобразованном графе

repair_chromosome(self, chromosome: list[int]) -> list[int] – пока включенные вершины не образуют клику, удаляет случайную вершину минимальной степени в подграфе. Возвращает новую хромосому

2. Класс *Parameters* – отвечает за хранение параметров генетического алгоритма, их загрузкой из файла.

Поля класса:

int population_size – Размер популяции

int max_generations – Максимальное количество итераций

int stagnation_limit – Количество итераций в застое

float max_mutation_prob_gene – Максимальная вероятность мутации 1 гена

float max_mutation_prob_chrom – Максимальная вероятность мутации хромосомы

float fitness_scaling_percent – Масштабирование приспособленности хромосом при селекции родителей (в процентах)

int max_crossover_points – Максимальное количество точек разреза

float decrease_percent – Процент уменьшения вероятности мутации и точек разреза

int decrease_step – Шаг (количество поколений) уменьшения точек разреза, вероятности мутации гена и хромосомы

Методы класса:

load_parameters_from_json(file_path: str) -> 'Parameters' – Загружает параметры генетического алгоритма из json-файла. Выбрасывает ошибку FileNotFoundError, если файл не существует или ValueError, если неверные значения параметров или отсутствуют ключи

3. Класс *Indididual* – отвечает за хранение конкретной особи, заданной хромосомой.

Поля класса:

list[int] chromosome – Бинарный вектор, задающий хромосому

float fitness – Размер клики

Методы класса:

evaluate(self) – вычисляет приспособленность особи как размер клики, считая, что заданная хромосома всегда задает клику

4. Класс *Population* – отвечает за работу с популяцией, хранит все необходимые параметры и имеет вспомогательные методы для работы с популяцией

Поля класса:

list[Individual] individuals – список особей

Individual best – лучшая особь в популяции

float avg_fitness – средняя приспособленность

Методы класса:

get_len(self) – Подсчет количества особоей в популяции

update_stats(self) – Пересчитывает статистики популяции

select_best(self, n: int) -> list[Individual] – Возвращает n лучших особей

get_fitnesses(self) -> list[float] – Возвращает список приспособленностей всех особей

add_individuals(self, new_individuals: list[Individual]) – Добавляет новых особей в популяцию

5. Класс *History* – отвечает за хранение промежуточных данных работы алгоритма для дальнейшей работы с ними, имеет вспомогательные методы

Поля класса:

list[int] generations – Список номеров поколений

list[float] best_fitness – Лучшие приспособленности на каждом поколении

list[float] avg_fitness – Средние приспособленности на каждом поколении

list[list[int]] best_solutions – Лучшие решения на каждом поколении (хромосомы)

Методы класса:

record(self, generation: int, population: Population) – Пересчитывает параметры популяции и сохраняет их в историю

plot_fitness(self) – Строит график динамики лучшей и средней приспособленности по поколениям

save_to_json(self, path: str) – Сохраняет историю работы алгоритма в результирующий json-файл

Пример работы алгоритма см в приложение А.

Реализация визуальной части GUI

Используемые библиотеки: tkinter (gui в целом), networkX (визуализация графа), matplotlib (график приспособленности), numpy.

В рамках доработки графического интерфейса была реализована двухоконная система на базе классов MainApp (главное окно) и MatrixWindow (окно матрицы смежности), обеспечивающая удобное взаимодействие с программой. Основное окно приложения, реализованное в классе MainApp, теперь содержит все ключевые элементы управления: панель параметров алгоритма, область визуализации графа через networkx, график изменения показателей приспособленности (FitnessPlotWidget), список найденных решений (SolutionListWidget) и элементы управления воспроизведением процесса поиска решений.

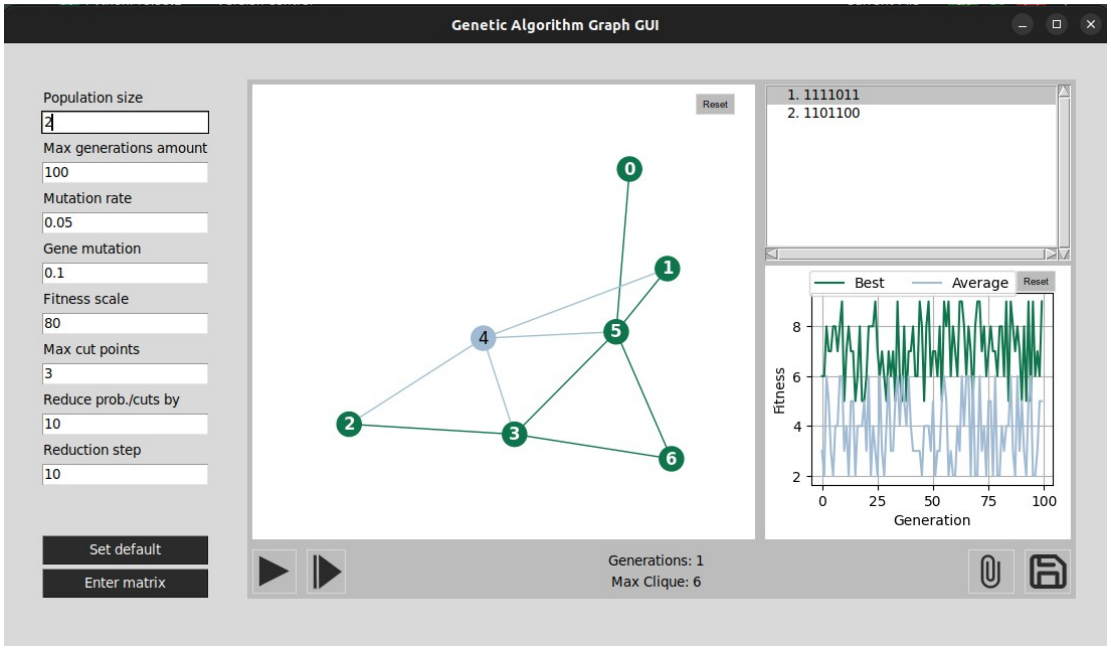
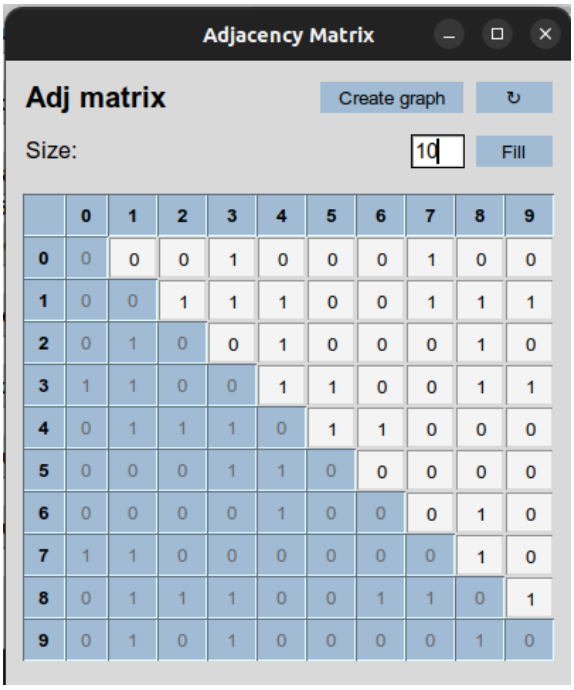


Рисунок 1 — Основное окно

Класс MatrixWindow отвечает за работу с матрицей смежности, предоставляя интерактивный редактор с автоматической проверкой вводимых значений через механизмы класса Validator. Пользователь может кликать по ячейкам таблицы, переключая значения между 0 и 1, при этом система автоматически обеспечивает симметричность матрицы благодаря синхронизации между элементами Button и Label.



	0	1	2	3	4	5	6	7	8	9
0	0	0	0	1	0	0	0	1	0	0
1	0	0	1	1	1	0	0	1	1	1
2	0	1	0	0	1	0	0	0	1	0
3	1	1	0	0	1	1	0	0	1	1
4	0	1	1	1	0	1	1	0	0	0
5	0	0	0	1	1	0	0	0	0	0
6	0	0	0	0	1	0	0	0	1	0
7	1	1	0	0	0	0	0	0	1	0
8	0	1	1	1	0	0	1	1	0	1
9	0	1	0	1	0	0	0	0	1	0

Рисунок 2 — Ввод матрицы

Интерактивные возможности интерфейса значительно расширены за счет класса ZoomableWidget, который обеспечивает функцию масштабирования с привязкой к положению курсора. Для навигации по объемным спискам решений в классе SolutionListWidget добавлены скроллбары. Элементы управления воспроизведением в MainApp позволяют как запускать алгоритм полностью, так и анализировать его работу пошагово.

Система активно проверяет корректность вводимых данных через методы класса Validator. Для параметров алгоритма установлены допустимые диапазоны значений, а для матрицы смежности автоматически проверяется соответствие требованиям (симметричность, нулевая диагональ). При обнаружении ошибок выводятся сообщения через messagebox.

Кратко перечислим ключевые методы для классов:

Класс Validator (Валидация данных)

Методы:

1. *validate_graph_exists(graph)* – Проверяет существование графа (не None). Возвращает True/False, показывает ошибку если графа нет.
2. *validate_graph_for_save(adj_matrix)* – Проверяет возможность сохранения матрицы (не None). Возвращает True/False, показывает ошибку если матрицы нет
3. *validate_matrix_size(size, max_size=15)* – Проверяет размер матрицы (не больше max_size). Возвращает скорректированный размер, показывает предупреждение.
4. *validate_cell_value(value)* – Проверяет значение ячейки (должно быть 0 или 1). Возвращает число или None при ошибке.
5. *validate_matrix(matrix)* – Проверяет матрицу на:
   - Квадратность
   - Симметричность
   - Нули на диагонали

   Возвращает (bool, сообщение)

Класс FileManager (Работа с файлами)

Методы:

1. *get_save_filename(title, defaultextension, filetypes)* – Открывает диалог сохранения файла. Возвращает путь к файлу.
2. *get_open_filename(title, filetypes)* – Открывает диалог выбора файла. Возвращает путь к файлу.
3. *save_matrix_to_file(filename, matrix)* – Сохраняет матрицу в текстовый файл. Возвращает True/False, показывает статус.
4. *load_matrix_from_file(filename)* – Загружает матрицу из файла. Возвращает (матрица, сообщение).

Класс UIManager (Создание UI-элементов)

Методы:

1. *create_frame(parent, **kwargs)* – Создает фрейм с настройками.
2. *create_label(parent, text, **kwargs)* – Создает текстовую метку.
3. *create_button(parent, text, command, **kwargs)* – Создает кнопку с обработчиком.
4. *create_entry(parent, **kwargs)* – Создает поле ввода.
5. *configure_grid_frame(frame, rows, cols)* – Настраивает сетку фрейма.
6. *create_table_header(parent, text, row, col, **kwargs)* – Создает заголовок таблицы

Класс MatrixGenerator (Генерация матриц)

Методы:

1. *generate_random_matrix(size)* – Генерирует случайную симметричную матрицу (0/1).
2. *matrix_to_text(matrix)* – Конвертирует матрицу в текстовый формат

Класс ZoomableWidget (Управление масштабированием)

Методы:

1. *_zoom_to_cursor(event, scale_factor)* – Масштабирует график/граф относительно курсора.
2. *_on_zoom(event)* – Обрабатывает событие колеса мыши (Windows/Mac).
3. *_on_zoom_linux(event, direction)* – Обрабатывает событие кнопок мыши (Linux).
4. *reset_zoom()* – Сбрасывает масштаб к исходному.
5. *save_original_limits()* – Сохраняет исходные границы осей.

Класс FitnessPlotWidget (График фитнеса)

Методы:

1. *draw_fitness(generations, best, avg)* – Рисует график эволюции фитнеса. Если данные пустые - рисует заглушку.
2. *reset_zoom()* – Сбрасывает масштаб (наследуется от ZoomableWidget)

Класс SolutionListWidget (Список решений)

Методы:

1. *set_example(size=5)* – Заполняет список тестовыми данными (для демо).

Класс MainApp (Главное окно)

Ключевые методы:

1. *update_graph(adj_matrix)* – Обновляет граф по новой матрице смежности.
2. *_draw_graph(highlight_clique=False)* – Отрисовывает граф с подсветкой клики.
3. *run_algorithm()* – Запускает генетический алгоритм (заглушка).
4. *save_graph(), load_adjacency_matrix()* – Работа с файлами через FileManager.

Класс MatrixWindow (Окно матрицы)

Ключевые методы:

1. *create_matrix_table()* – Создает таблицу для редактирования матрицы
2. *toggle_cell(i, j)* – Переключает значение ячейки (0/1)
3. *random_matrix()* – Генерирует случайную матрицу
4. *create_graph()* – Применяет матрицу к главному окну

ПРИЛОЖЕНИЕ А

ТЕСТИРОВАНИЕ АЛГОРИТМА

graph1.json:


```
{
    "0": [1, 2, 3, 4, 5, 6, 7, 8, 9, 10],
    "1": [0, 2, 3, 4, 5, 6, 7, 11, 12],
    "2": [0, 1, 3, 4, 5, 6, 7, 13, 14],
    "3": [0, 1, 2, 4, 5, 6, 7, 15],
    "4": [0, 1, 2, 3, 5, 6, 7, 16],
    "5": [0, 1, 2, 3, 4, 6, 7, 17],
    "6": [0, 1, 2, 3, 4, 5, 7, 18],
    "7": [0, 1, 2, 3, 4, 5, 6, 19],
    "8": [0, 9, 10, 11, 20, 21],
    "9": [0, 8, 12, 13, 20, 21],
    "10": [0, 8, 14, 15, 20],
    "11": [1, 8, 16, 17, 20],
    "12": [1, 9, 18, 19, 20],
    "13": [2, 9, 16, 17, 20],
    "14": [2, 10, 18, 19, 20],
    "15": [3, 10, 16, 18, 20],
    "16": [4, 11, 13, 15, 20],
    "17": [5, 11, 13, 19, 20],
    "18": [6, 12, 14, 15, 20],
    "19": [7, 12, 14, 17, 20],
    "20": [8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19],
    "21": [8, 9, 22],
    "22": [21, 23],
    "23": [22, 24],
    "24": [23, 25],
    "25": [24, 26],
    "26": [25]
}
```


Вывод при тесте:

```
Начало работы генетического алгоритма
Размер графа: 27 вершин
Параметры алгоритма:
  Размер популяции: 5
  Макс. поколений: 100
  Лимит застоя: 20
  Начальные точки кроссовера: 5

Начальная популяция:
  Особа 1: 01111111100000000000000000 (размер: 8)
  Особа 2: 01111111100000000000000000 (размер: 8)
  Особа 3: 10000000000010000010000000 (размер: 3)
  Особа 4: 01111111100000000000000000 (размер: 8)
  Особа 5: 10000000001001000000000000 (размер: 3)

Поколение 1:
  Лучший размер клики: 8
  Застой: 1 / 20
  Параметры: точки=5, мутация хромосомы=0.300, мутация гена=0.100
  Текущая популяция:
```

```
    Особа 1: 01111111100000000000000000 (размер: 8)
    Особа 2: 10000000000010000010000000 (размер: 3)
    Особа 3: 00000101100000000000000000 (размер: 3)
    Особа 4: 10000000001001000000000000 (размер: 3)
    Особа 5: 10000000000000100010000000 (размер: 3)
  Лучшая клика: вершины [0, 1, 2, 3, 4, 5, 6, 7]

Поколение 2:
  Лучший размер клики: 8
  Застой: 2 / 20
  Параметры: точки=5, мутация хромосомы=0.300, мутация гена=0.100
  Текущая популяция:
    Особа 1: 01111111100000000000000000 (размер: 8)
    Особа 2: 10000000000010000010000000 (размер: 3)
    Особа 3: 00000101100000000000000000 (размер: 3)
    Особа 4: 10000000001001000000000000 (размер: 3)
    Особа 5: 10000000000000100010000000 (размер: 3)
  Лучшая клика: вершины [0, 1, 2, 3, 4, 5, 6, 7]

Поколение 3:
  Лучший размер клики: 8
  Застой: 3 / 20
  Параметры: точки=5, мутация хромосомы=0.300, мутация гена=0.100
  Текущая популяция:
    Особа 1: 01111111100000000000000000 (размер: 8)
    Особа 2: 10000000000010000010000000 (размер: 3)
    Особа 3: 00000101100000000000000000 (размер: 3)
    Особа 4: 10000000001001000000000000 (размер: 3)
    Особа 5: 00000000000000000000000010 (размер: 1)
  Лучшая клика: вершины [0, 1, 2, 3, 4, 5, 6, 7]

…
Сокращение вывода
…

Поколение 17:
  Лучший размер клики: 8
  Застой: 17 / 20
  Параметры: точки=4, мутация хромосомы=0.270, мутация гена=0.090
  Текущая популяция:
    Особа 1: 01111111100000000000000000 (размер: 8)
    Особа 2: 10000000000010000010000000 (размер: 3)
    Особа 3: 00000101100000000000000000 (размер: 3)
    Особа 4: 01000000000100000000000000 (размер: 2)
    Особа 5: 10000000001001000000000000 (размер: 3)
  Лучшая клика: вершины [0, 1, 2, 3, 4, 5, 6, 7]

Поколение 18:
  Лучший размер клики: 8
  Застой: 18 / 20
  Параметры: точки=4, мутация хромосомы=0.270, мутация гена=0.090
  Текущая популяция:
    Особа 1: 01111111100000000000000000 (размер: 8)
    Особа 2: 10000000000010000010000000 (размер: 3)
    Особа 3: 00000101100000000000000000 (размер: 3)
    Особа 4: 01000000000100000000000000 (размер: 2)
    Особа 5: 10000000001001000000000000 (размер: 3)
  Лучшая клика: вершины [0, 1, 2, 3, 4, 5, 6, 7]
```

```
Поколение 19:
  Лучший размер клики: 8
  Застой: 19 / 20
  Параметры: точки=4, мутация хромосомы=0.270, мутация гена=0.090
  Текущая популяция:
    Особа 1: 01111111100000000000000000 (размер: 8)
    Особа 2: 10000000000010000100000000 (размер: 3)
    Особа 3: 00000101100000000000000000 (размер: 3)
    Особа 4: 01000000000100000000000000 (размер: 2)
    Особа 5: 10000000001001000000000000 (размер: 3)
  Лучшая клика: вершины [0, 1, 2, 3, 4, 5, 6, 7]

Поколение 20:
  Лучший размер клики: 8
  Застой: 20 / 20
  Параметры: точки=3, мутация хромосомы=0.240, мутация гена=0.080
  Текущая популяция:
    Особа 1: 01111111100000000000000000 (размер: 8)
    Особа 2: 10000000000010000100000000 (размер: 3)
    Особа 3: 00000101100000000000000000 (размер: 3)
    Особа 4: 01000000000100000000000000 (размер: 2)
    Особа 5: 10000000001001000000000000 (размер: 3)
  Лучшая клика: вершины [0, 1, 2, 3, 4, 5, 6, 7]


Результат работы алгоритма:
Найдена клика размера 8
Вершины в клике: [0, 1, 2, 3, 4, 5, 6, 7]
Всего поколений: 20
Клика валидна
Причина остановки: Превышен лимит застоя
```