САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ФАКУЛЬТЕТ ПРИКЛАДНОЙ МАТЕМАТИКИ – ПРОЦЕССОВ УПРАВЛЕНИЯ

ОТЧЕТ

по лабораторной работе №9

по дисциплине «Алгоритмы и структуры данных»

на тему «Алгоритм имитации отжига»

Выполнила
студентка 2 курса
группы 21-Б15.ПУ
Павлова Ксения Андреевна

Преподаватель
Дик Александр Геннадьевич

Санкт-Петербург
2023

СОДЕРЖАНИЕ

Цель: исследование особенностей решения задачи о коммивояжере с помощью алгоритма имитации отжига и сравнение алгоритма с методом ближайшего соседа.

Задачи:

1. Формализовать задачу о коммивояжере с помощью алгоритма имитации отжига.
2. Подготовить контрольный пример, используя взвешенный граф.
3. Найти кратчайший гамильтонов цикл.
4. Сравнить решение задачи о коммивояжере с помощью алгоритма ближайшего соседа (с результатами лабораторной работы №8).
5. Написать пользовательский интерфейс для удобства пользования алгоритмом.

Ход работы.

Краткое описание алгоритма имитации отжига.

Алгоритм имитации отжига (Simulated annealing) — алгоритм решения различных оптимизационных задач. Он основан на моделировании реального физического процесса, который происходит при кристаллизации вещества из жидкого состояния в твёрдое, в том числе при отжиге металлов.

Целью алгоритма является минимизация некоторого функционала. В процессе работы алгоритма хранится текущее решение, которое является промежуточным результатом. А после работы алгоритма оно и будет ответом.

Описание схемы пошагового выполнения алгоритма.

Шаги алгоритма:

1. Выбор начального решения и начальной температуры.

Хорошей стратегий является случайный выбор начального решения. Также в качестве начального решения можно предложить решение, полученное другими методами. Это предоставляет алгоритму базу, на основании которой он будет строить более оптимальное решение.

Выбор начальной и пороговой температуры следует производить экспериментально. Естественными рекомендациями могут служить выбор пороговой температуры близкой к нулю, а начальной достаточно высокой.

2. Оценка начального решения.

Этот этап полностью завит от специфики задачи. Единственным требованием является получение в качестве оценки одного вещественного числа, которое будет характеризовать оптимальность предлагаемого решения. Это число в алгоритме имитации отжига принято называть энергией. Если выбор такого числа является затруднительным, то, возможно, стоит отказаться от использования предлагаемого метода.

3. Основной шаг алгоритма.

Основной шаг при некоторой температуре повторяется несколько раз. Возможно, что один раз. Также возможен вариант с зависимостью числа повторов от температуры.

1. Случайное изменение текущего решения.

Этот этап сильно зависит от специфики задачи. Однако, изменение стоит производить локальные. Например, для задачи коммивояжера, хорошей стратегий будет обмен, в текущем порядке следования городов, двух случайных городов местами. В результате изменения у нас будет два решения: текущее и измененное.

2. Оценка измененного решения.
3. Критерий допуска.

Для определенности будем считать, что оптимизация заключается в минимизации энергии. В большинстве случаев этот подход справедлив. Критерий допуска заключается в проверке и возможной замене текущего решения измененным.

Если измененное решение имеет меньшую энергию, то оно принимается за текущее. Если же измененное решение имеет большую энергию, то оно принимается с вероятностью $P = e^{(-\frac{\delta E}{T})}$, где

- P — вероятность принять измененное решение,
- δE — модуль разности между энергией оптимального решения и энергий измененного решения,
- T — текущая температура.

4. Уменьшение температуры.

Важной частью алгоритма является уменьшение температуры. При большой температуре вероятность выбора менее оптимального решения высока. Однако, в процессе работы алгоритма температура снижается, и вероятность выбора менее оптимального решения снижается.

Выбор способа уменьшения температуры может быть различным и выбирается экспериментально. Главное, чтобы температура монотонно убывала к нулю. Хорошей стратегий является умножение на каждом шаге температуры на некоторый коэффициент немного меньший единицы.

5. Если температура больше некоторого порога, то перейти к пункту 3.

Формализация задачи.

Для решения поставленной задачи был выбран язык программирования Python. Визуализация выполнялась с помощью библиотеки TKinter.

1. Выбор начального решения и начальной температуры.

Значения начальной «initial_temp» и пороговой «stopping_temp» температур задаются в интерфейсе пользователя. По умолчанию начальная температура равна 1000, пороговая – 10^{-8}.

Начальное решение «current_order» формируется случайным образом.

2. Оценка решений.

Оценка решений происходит с помощью функции «total_distance», которая вычисляет общее расстояние между городами в порядке, заданном входным параметром «order». Эта функция используется для оценки текущего порядка городов «current_order» и нового порядка городов «new_order», полученного в результате перестановки двух случайных городов.

3. Основной шаг алгоритма.

    1. Случайное изменение текущего решения.

    Случайное изменение текущего решение происходит путём перестановки двух случайных городов. В результате изменения у нас будет два решения: текущее «current_order» и измененное «new_order».

    2. Критерий допуска.

    Если новый порядок лучше (т.е. общее расстояние между городами меньше), то он принимается как текущий порядок. Если новый порядок хуже, то он может быть принят с некоторой вероятностью «prob», которая зависит от разницы в расстоянии между текущим и новым порядком

«delta», а также от текущей температуры. Чем выше температура, тем выше вероятность принять худший порядок. По мере уменьшения температуры вероятность принять худший порядок уменьшается, что позволяет алгоритму сходиться к лучшему решению.

4. Уменьшение температуры.

   На каждой итерации значение текущей температуры умножается на коэффициент (1 - cooling_rate), где «cooling_rate» – это параметр, задающий скорость охлаждения. Скорость охлаждения задаётся в пользовательском интерфейсе и по умолчанию равна 0,003. Чем меньше значение «cooling_rate», тем медленнее уменьшается температура.

5. При достижении заданного значения «stopping_temp» процесс остановится, и лучший найденный порядок городов будет возвращен в качестве решения задачи коммивояжера. В противном случае алгоритм вернётся к шагу 3.

Листинг программы.

Алгоритм имитации отжига.

```
#Блок подключения необходимых библиотек
import random
import math

#Поиск расстояния между двумя городами
def distance(city1, city2):
    for i in mass_edges:
        if city1 in i and city2 in i:
            return i[2]
    return

#Оценка решений
def total_distance(cities, order):
    dist = 0
    for i in range(len(order)):
        dist += distance(cities[order[i]], cities[order[(i + 1)
        % len(order)]])
    return dist

#Основа алгоритма имитации отжига
def solve_tsp_with_simulated_annealing(mass):
    #Инициализация параметров
    k = 0
    global mass_edges
```

```
    canvas = mass[0]
    initial_temp = int(mass[1].get())
    stopping_temp = float(mass[2].get())
    cooling_rate = float(mass[3].get())
    canvas.best_edges_delete()
    canvas.edges_mass =
    canvas.table.check_data(canvas.edges_mass)
    verticles, mass_edges = canvas.circle_mass,
canvas.edges_mass
    cities = []
    for i in range(len(verticles)):
        cities.append(verticles[i][0])
    #Создание начального решения (случайным образом)
    current_order = list(range(len(cities)))
    random.shuffle(current_order)
    current_temp = initial_temp

    best_order = current_order[:]
    best_dist = total_distance(cities, current_order)
    #Пока текущая температура больше пороговой
    while current_temp > stopping_temp:
        k+=1
        #Изменение решения путём обмена двух случайных городов
        местами
        new_order = current_order[:]
        i, j = random.sample(range(len(cities)), 2)
        new_order[i], new_order[j] = new_order[j], new_order[i]
        #Оценка изменённого решения
        current_dist = total_distance(cities, current_order)
        new_dist = total_distance(cities, new_order)

        if new_dist < current_dist: #Если новый порядок лучше
            current_order = new_order[:]
            if new_dist < best_dist:
                best_order = new_order[:]
                best_dist = new_dist
        else: #Если новый порядок хуже, то замена с некоторой
        вероятностью
            delta = new_dist - current_dist
            prob = math.exp(-delta / current_temp)
            if random.random() < prob:
                current_order = new_order[:]
    #Уменьшение температуры
        current_temp *= 1 - cooling_rate
    #Пересвоение лучших величин
    best_order.append(best_order[0])
    canvas.best_weight = best_dist
    canvas.best_way = best_order
    canvas.all_edges_hidden(True)
    canvas.counter = k
    canvas.draw_best_way()
    return canvas
```

```
#Массив рёбер
mass_edges = []
```

Визуализация.

```
#Блок подключения необходимых библиотек
import tkinter as tk
from tkinter import ttk
from tkinter import simpledialog

class myWindow:
    def __init__(self, root, color):
        self.root = root
        self.color = color
        #Создание главных фреймов
        self.left_frame = tk.Frame(self.root, height=250,
width=500)
        self.right_frame = tk.Frame(self.root, height=250,
width=500)
        self.left_frame.grid(row=0, column=0, sticky="nsew")
        self.right_frame.grid(row=0, column=1, sticky="nsew")
        #Задание пропорций растяжения для колонок
        self.root.columnconfigure(0, weight=1)
        self.root.columnconfigure(1, weight=1)
        #Задание обработчика изменения размера окна
        self.root.bind("<Configure>", self.on_resize)
        #Создание фреймов в левом фрейме
        self.top_left_frame = tk.LabelFrame(self.left_frame,
text='Input area', bg=self.color, height=125, width=500)
        self.middle_left_frame = tk.LabelFrame(self.left_frame,
text='Table area', bg=self.color, height=125, width=500)
        self.bottom_left_frame = tk.LabelFrame(self.left_frame,
text='Result area', bg=self.color, height=125, width=500)
        self.top_left_frame.pack(side="top", fill="both",
expand=True)
        self.middle_left_frame.pack(side="top", fill="both",
expand=True)
        self.bottom_left_frame.pack(side="bottom", fill="both",
expand=True)
        #Создание фреймов в правом фрейме
        self.top_right_frame = tk.LabelFrame(self.right_frame,
text='Canvas area', bg=self.color, height=125, width=500)
        self.top_right_frame.pack(side="top", fill="both",
expand=True)

    #Функция запуска
    def on_resize(self, event):
        #Получение текущей ширины окна
        width = self.root.winfo_width()
        #Вычисление новой ширины главных фреймов
        left_width = int(width / 3)
        right_width = width - left_width
        #Изменение ширины колонок
```

```
        self.root.columnconfigure(0, minsize=left_width)
        self.root.columnconfigure(1, minsize=right_width)

#Общее описание кнопок
class Button:
    def __init__(self, root,text,command,arg):
        self.arg = arg
        self.command = command
        self.root = root
        self.text = text
        button = tk.Button(self.root, text=self.text,
command=lambda: self.command(self.arg))
        button.pack()

#Описание полей с вводом значений
class labeledSpinbox():
    def __init__(self, root,label_text, spinbox_from,
spinbox_to, spinbox_default,spinbox_step):
        self.root = root
        self.label_text = label_text
        self.spinbox_from = spinbox_from
        self.spinbox_to = spinbox_to
        self.spinbox_default = spinbox_default
        self.spinbox_step = spinbox_step
        #Создание виджета Label
        self.widget_frame = tk.Frame(self.root)
        self.widget_frame.pack(side=tk.TOP)
        self.label = tk.Label(self.widget_frame,
text=self.label_text,font=("Arial", 10))
        self.label.pack(side=tk.LEFT)
        # Создание виджета Spinbox
        self.spinbox = tk.Spinbox(self.widget_frame,
from_=self.spinbox_from, to=self.spinbox_to, width=10,
increment=self.spinbox_step,font=("Arial", 10))
        self.spinbox.delete(0, tk.END)  #Удаление стандартного
значения
        self.spinbox.insert(0, spinbox_default)  #Установка
стандартного значения
        self.spinbox.pack(side=tk.LEFT)

    def get(self):
        return self.spinbox.get()
#Создание холстов и рисование графов
class canvas:
    def __init__(self, root,table,result):
        self.root = root
        self.table = table
        self.result = result
        self.canvas = tk.Canvas(self.root, width=400,
height=400, bg='white')
        self.canvas.pack(fill="both", expand=True)
        self.circle_count = 0
```

```
        self.circle_mass = []
        self.edges_mass = []
        self.best_way = []
        self.best_weight=0
        self.counter = 0
        #Привязка функции рисования к событию нажатия мыши
        self.canvas.bind("<Button-1>", self.draw_circle)
        self.canvas.bind("<Button-3>", self.circle_delete)
        self.canvas.bind("<Button-2>", self.show_data)

    #Функция рисования круга на холсте
    def show_data(self,event):
        print(self.edges_mass)
        print(self.circle_mass)
    def draw_circle(self, event):
        self.best_edges_delete()
        self.all_edges_hidden(False)
        x, y = event.x, event.y
        r = 20
        overlapping_circles = self.canvas.find_enclosed(x - r-
        25, y - r-25, x + r+25, y + r+25)  #поиск
перекрывающихся кругов
        if not overlapping_circles:  #если нет пересечений
            #Добавление новой вершины
            self.circle_mass.append([self.circle_count,
            [int(x),int(y)]])
            self.canvas.create_oval(x - r, y - r, x + r, y + r,
            fill='red',tags='oval')
            self.canvas.create_text(x, y,
            text=str(self.circle_count), font=('Arial', 12,
            'bold'),tags='text')
            self.edges_mass =
            self.table.check_data(self.edges_mass)
            self.draw_edges(self.circle_count)
            self.circle_count += 1
            self.table.update_data(self.edges_mass)
        self.canvas.tag_raise('oval')
        self.canvas.tag_raise('text')

    #Рисование рёбер
    def draw_edges(self, id):
        l = len(self.circle_mass)
        for i in range(l-1):
            x1, y1 = self.circle_mass[l-1][1]
            x2, y2 = self.circle_mass[i][1]
            id1 = self.circle_mass[l-1][0]
            id2 = self.circle_mass[i][0]
            self.canvas.create_line(x1, y1, x2, y2, width=1,
            fill="black",tags='lines')
            weight = ((x2 - x1)**2 + (y2 - y1)**2)**0.5
            self.edges_mass.append([id1,id2,int(weight)])
```

```
    #Удаление вершины
    def circle_delete(self, event):
        self.edges_mass = self.table.check_data(self.edges_mass)
        x = event.x
        y = event.y
        r = 20
        objects = self.canvas.find_overlapping(x-r,y-r,x+r,y+r)
        for obj in objects:
            if self.canvas.type(obj) == 'oval':
                coords = self.canvas.coords(obj)
                circle_x = (coords[0] + coords[2]) // 2
                circle_y = (coords[1] + coords[3]) // 2
                for oval in self.circle_mass:
                    if oval[1][0] == circle_x and oval[1][1] ==
                    circle_y:
                        self.delId = oval[0]
                        self.circle_mass.remove(oval)
                        self.edges_delete()
            self.canvas.delete(obj)
        self.table.update_data(self.edges_mass)

    #удаление ребра
    def edges_delete(self):
        l = len(self.edges_mass)
        self.edges_mass = [inner_lst for inner_lst in
        self.edges_mass if self.delId not in inner_lst]

    #скрытие и показ рёбер
    def all_edges_hidden(self,Bool):
        items = self.canvas.find_withtag("lines")
        for item in items:
            if Bool:
                self.canvas.itemconfig(item, state='hidden')
            else:
                self.canvas.itemconfig(item, state='normal')

    #Удаление рёбер найденного пути
    def best_edges_delete(self):
        items = self.canvas.find_withtag("bestlines")
        for item in items:
          self.canvas.delete(item)

    #Отрисовка найденного пути
    def draw_best_way(self):
        self.result.update_data_res(self.best_way,
        self.best_weight,self.counter)
        coordinates = []
        for i in self.best_way:
            for j in self.circle_mass:
                if j[0] == i:
                    coordinates.append(j[1])
```

```
        coordinates.append(coordinates[0])
        for i in range(len(coordinates)-1):
            self.canvas.create_line(coordinates[i][0],
coordinates[i][1], coordinates[i+1][0], coordinates[i+1][1],
width=2, fill="red",tags='bestlines')
        self.canvas.tag_raise('text')

    #Очистка холста
    def clear_canvas(self,bool):
        self.canvas.delete("all")
        self.edges_mass = []
        self.circle_mass = []
        self.best_way = []
        self.best_weight = 0
        self.circle_count=0
        self.counter = 0
        self.table.update_data(self.edges_mass)
        self.result.update_data_res(self.best_way,
        self.best_weight,self.counter)

#Задание таблицы смежности
class myTable:
    def __init__(self,root,headers):
        self.root = root
        self.headers = headers
        self.changed_mass = []
        self.changed_val = []
        self.table = ttk.Treeview(self.root,
        columns=self.headers, show='headings')
        for header in self.headers:
            self.table.heading(header, text=header.title())
        for col in self.table["columns"]:
            self.table.column(col, width=100)
        self.table.pack(side=tk.TOP, fill=tk.X, expand=1)
        self.table.bind("<Double-1>", self.update_cell)

    #Обновление данных
    def update_data(self,mass):
        self.table.delete(*self.table.get_children())
        for row in mass:
            self.table.insert("", "end", values=row)
    #Проверка данных
    def check_data(self,list1):
        if len(self.changed_mass):
            for i in range(len(list1)):
                if list1[i] in self.changed_mass:
                    list1[i][2] = self.changed_val.pop(0)
            self.changed_mass = []
            self.changed_val = []
        return list1
    #Обновление значение ячеек
    def update_cell(self,event):
```

```
        #Получение ссылки на таблицу и на выбранную строку
        self.table = event.widget
        self.item = self.table.selection()[0]
        #Получение индекса выбранной колонки и имени выбранного
столбца
        self.column = self.table.identify_column(event.x)
        self.column_name =
self.table.heading(self.column)['text']
        #Получение старого значения ячейки и запрос у
пользователя нового значения
        self.old_value = self.table.set(self.item, self.column)
        self.id1 = self.table.set(self.item, '#1')
        self.id2 = self.table.set(self.item, '#2')
        self.new_value = simpledialog.askstring('Изменение
значения', f'Введите новое значение для
{self.column_name}:',initialvalue=self.old_value)
        #Обновление значения ячейки, если пользователь ввёл
новое значение
        if self.new_value:
            self.changed_mass.append([int(self.id1),
            int(self.id2),int(self.old_value)])
            self.changed_val.append(int(self.new_value))
            self.table.set(self.item, self.column,
            self.new_value)

#Задание блока вывода результатов
class myResult:
    def __init__(self,root):
        self.root = root
        self.visited = []
        self.best_weight = 0
        self.counter = 0
        self.label = tk.Label(self.root, text=f"Путь:
        {self.visited}")
        self.label2 = tk.Label(self.root, text=f"Общий вес пути:
        {self.best_weight}")
        self.label3 = tk.Label(self.root, text=f"Количество
        поколений: {self.counter}")
        self.label.pack()
        self.label2.pack()
    #Обновление результатов
    def update_data_res(self,vis,wg,counter):
        self.visited = vis
        self.best_weight = wg
        self.counter = counter
        self.label.config(text=f"Путь: {self.visited}")
        self.label2.config(text=f"Общий вес пути:
        {self.best_weight}")
        if self.counter:
            self.label3.config(text=f"Количество поколений:
            {self.counter}")
            self.label3.pack()
```

Подключение визуализации и основного алгоритма.

```
#Блок подключения необходимых библиотек
import tkinter as tk
from tkinter import ttk, messagebox
import gui
from alg import nearest_neighbor

#Функция закрытия программы
def on_closing():
    if messagebox.askokcancel("Выход", "Вы действительно хотите
выйти?"): root.destroy()

#Создание окна с вкладками и фреймов
root = tk.Tk()
root.geometry("1000x500")
root.protocol("WM_DELETE_WINDOW", on_closing)

notebook = ttk.Notebook(root)
notebook.pack(fill='both', expand=True)

tab1 = tk.Frame(notebook)
tab2 = tk.Frame(notebook)
tab3 = tk.Frame(notebook)
notebook.add(tab1, text='Метод ближайшего соседа')

#Запуск алгоритма имитации отжига
annealing_window = gui.myWindow(tab2,'#d7e8d5')
annealing_table =
gui.myTable(annealing_window.middle_left_frame,['Вершина 1',
'Вершина 2', 'Вес'])
annealing_result =
gui.myResult(annealing_window.bottom_left_frame)
annealing_canvas =
gui.canvas(annealing_window.top_right_frame,annealing_table,anne
aling_result)
annealing_stemp =
gui.labeledSpinbox(annealing_window.top_left_frame,"Начальная
температура",10,1000,1000,10)
annealing_fTemp =
gui.labeledSpinbox(annealing_window.top_left_frame,"Конечная
температура",0,1000,1e-8,1)
annealing_coolVal =
gui.labeledSpinbox(annealing_window.top_left_frame,"Коэффициент
охлаждения",0,1,0.003,0.001)
annealing_startButton =
gui.Button(annealing_window.top_left_frame,'Начать
алгоритм',annealing.solve_tsp_with_simulated_annealing,[annealin
g_canvas,annealing_stemp,annealing_fTemp,annealing_coolVal])
annealing_clearButton =
gui.Button(annealing_window.top_left_frame,'Стереть
данные',annealing_canvas.clear_canvas,True)
```

```
root.mainloop()
```

Спецификация программы.

Пользовательский интерфейс (рис. 1)

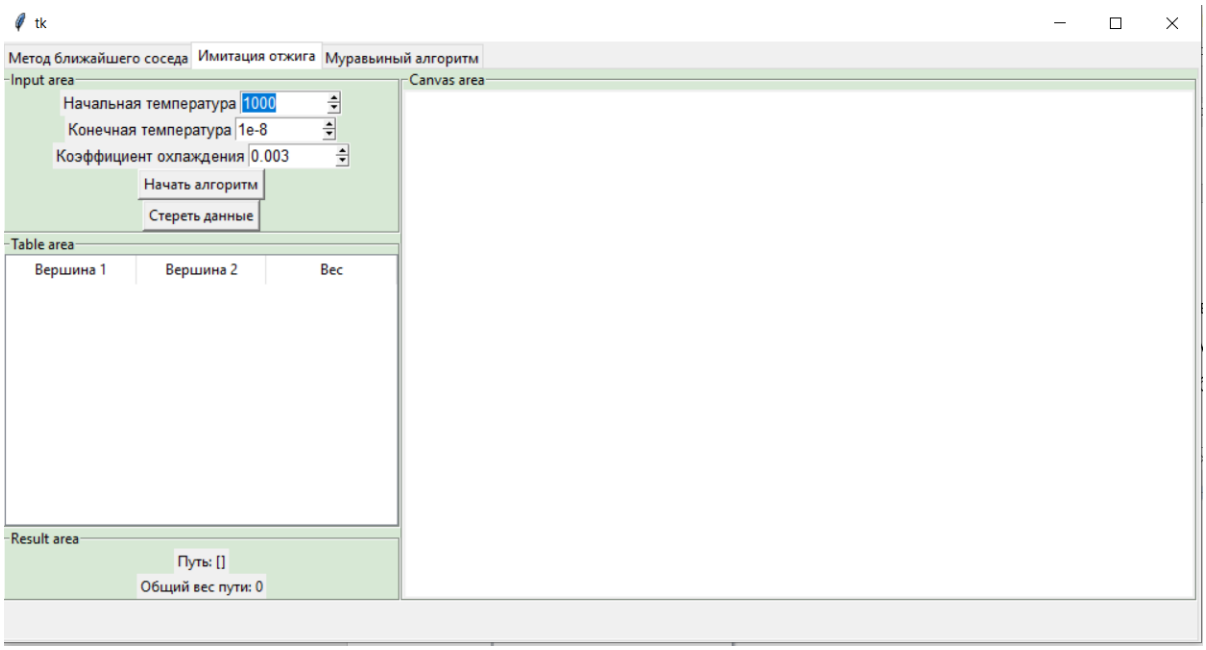

Рис. 1. Пользовательский интерфейс

Для создания графа используется правый фрейм. При нажатии ЛКМ создаётся вершина, которая соединяется со всеми остальными вершинами. Каждому ребру задаётся вес в соответствии с расстоянием до других вершин. Чтобы удалить вершину или ребро нужно нажать на соответствующий элемент ПКМ. Чтобы запустить алгоритм надо нажать кнопку «Начать алгоритм» в левой верхней части окна. Под ней располагается кнопка «Стереть данные», позволяющая очистить область рисования. Также в середине левого фрейма находится таблица со списком рёбер и весами. У каждого ребра можно вручную изменить вес, нажав на него в таблице дважды ЛКМ. В нижней части левого фрейма находится область вывода результата. При нажатии на знак крестика в правом верхнем углу осуществляется выход из программы, который нужно подтвердить. Если не ввести граф, то алгоритм выдаст ошибку «ValueError: Sample larger than population or is negative».

Контрольный пример.

Тестирование программы производилось на графе, представленном на рис. 2 с случайно заданными весами рёбер и параметрами по умолчанию.

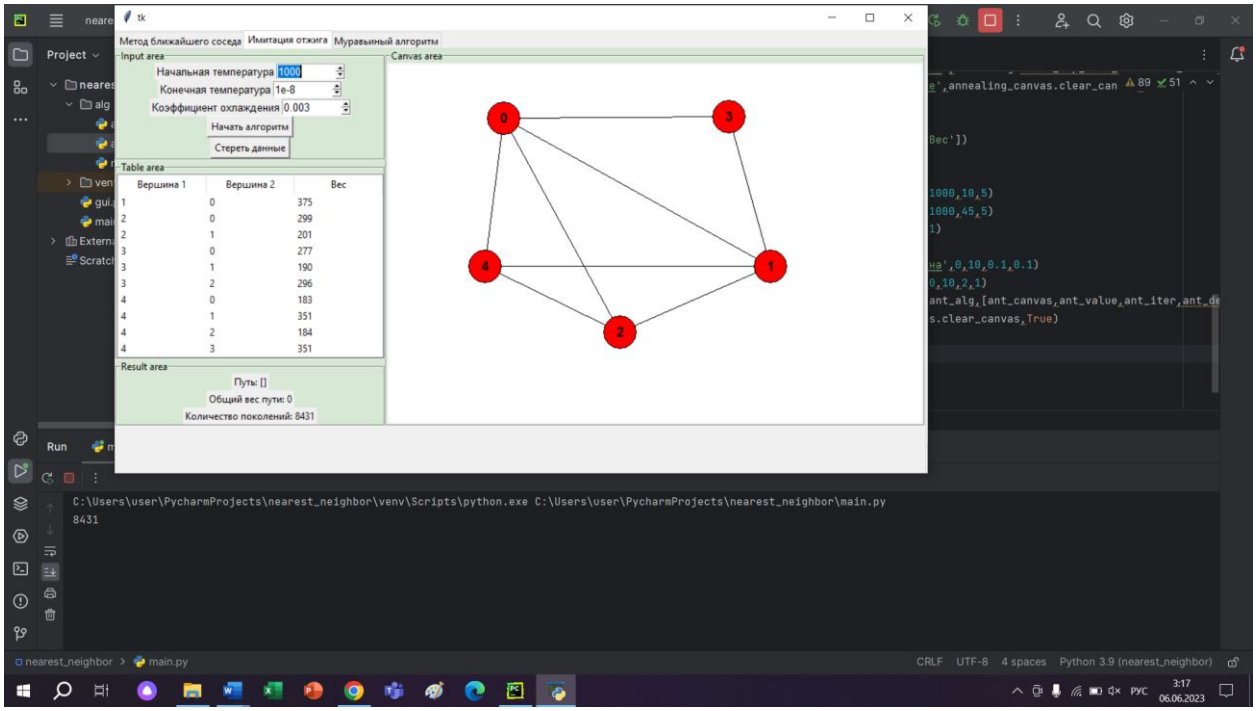

Рис. 2. Начало работы алгоритма

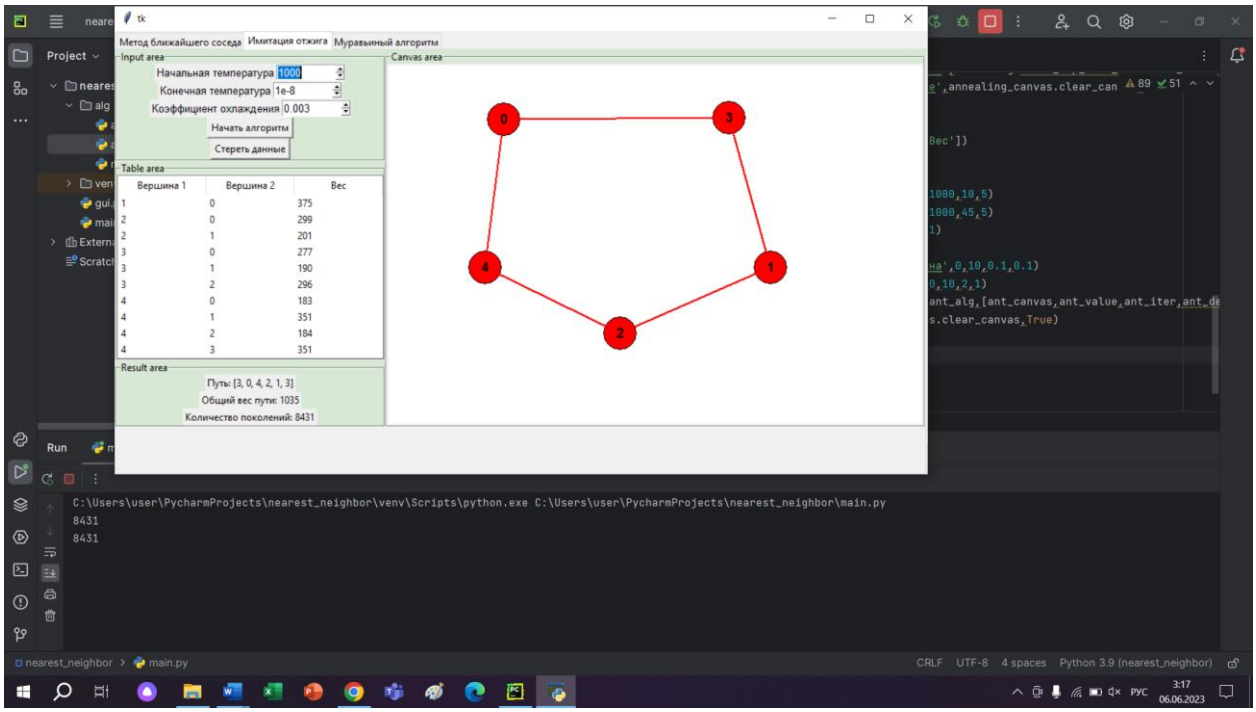

Рис. 3. Окончание работы алгоритма

Таким образом, найден кратчайший гамильтонов цикл, представленный на рис. 3, путь цикла по вершинам: 3, 0, 4, 2, 1, 3.

Анализ результатов работы алгоритма.

Анализ результатов работы алгоритма проводился с помощью тестового графа, представленного на рис. 4.

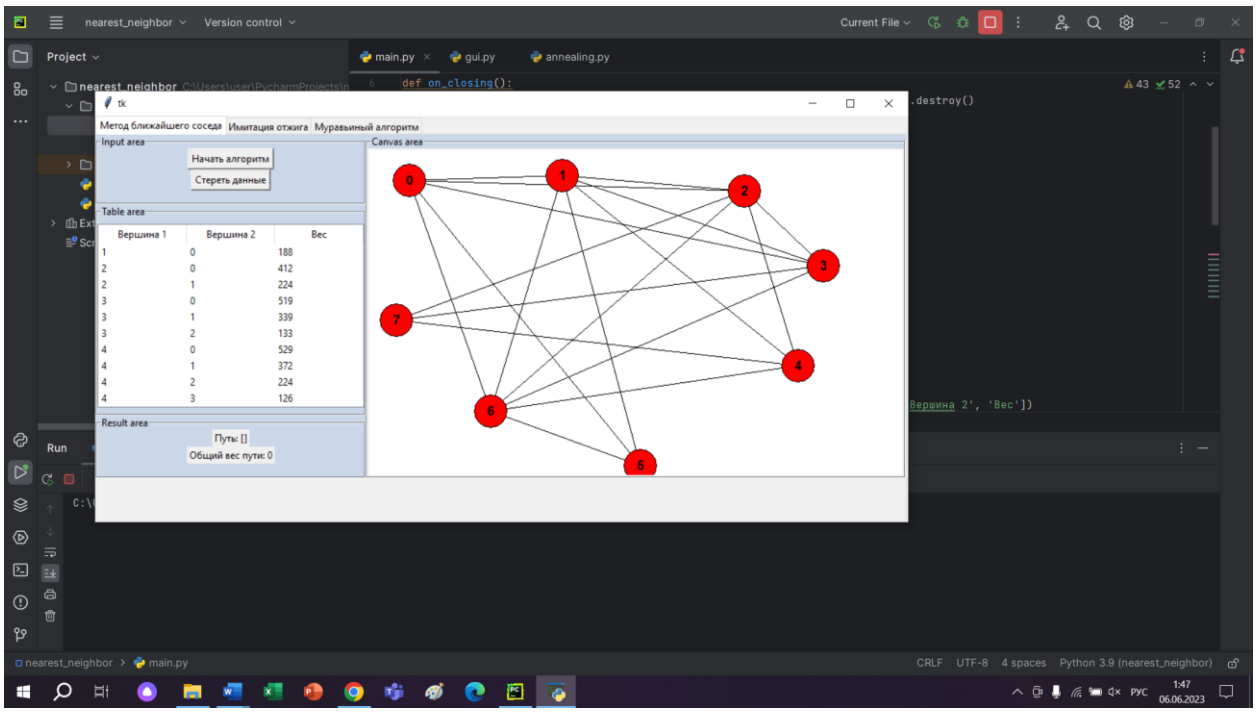

Рис. 4. Тестовый граф

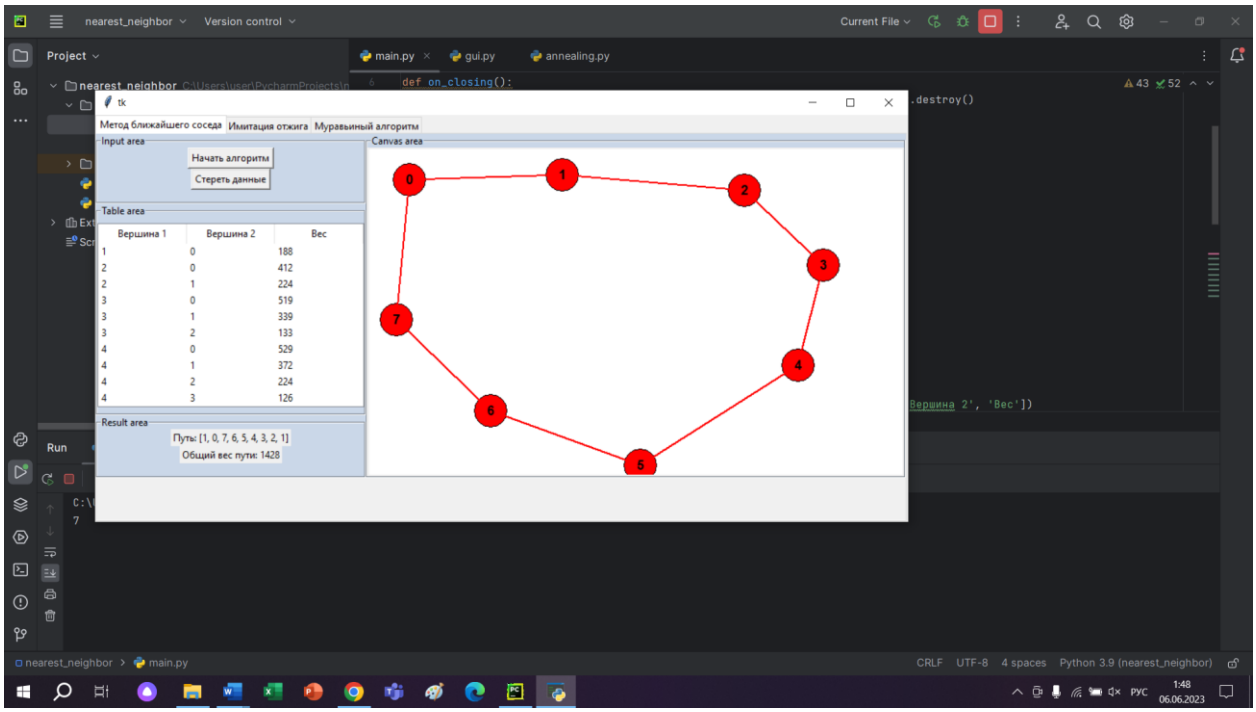

Рис. 5. Результат работы метода ближайшего соседа.

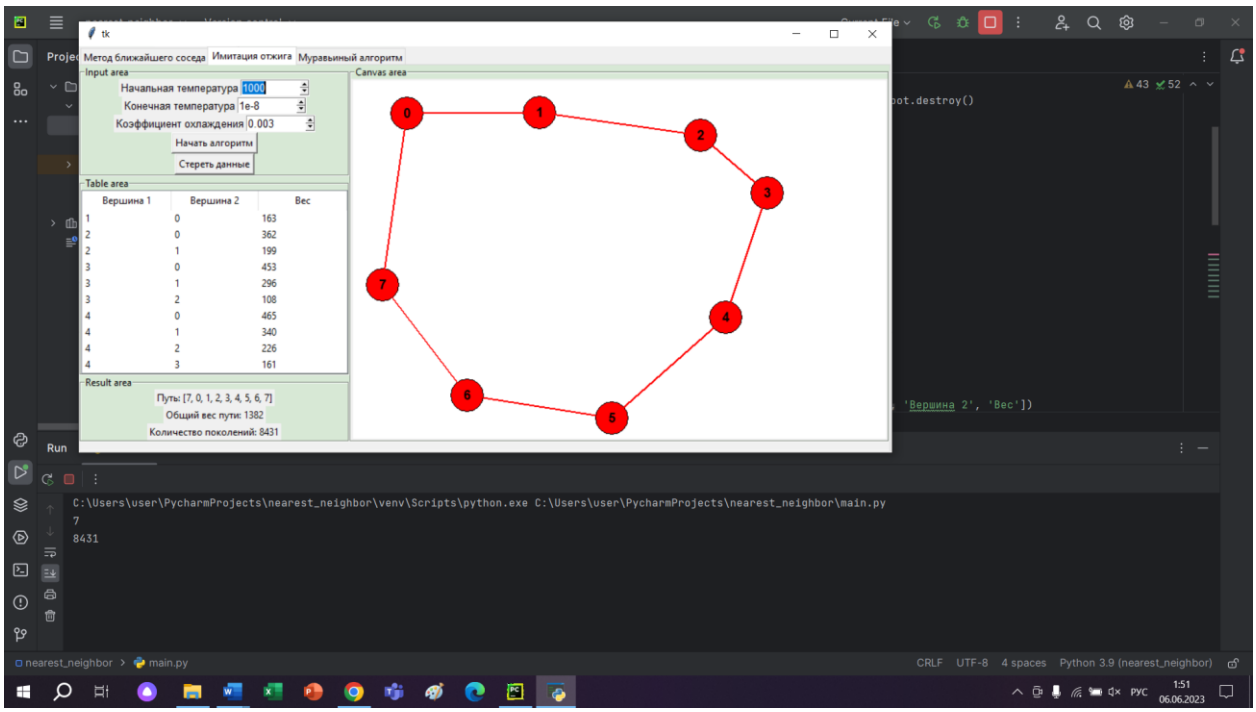

Рис. 6. Результат работы алгоритма имитации отжига

Анализ результатов работы алгоритма показал, что алгоритм имитации отжига в большем количестве случаев находит оптимальное решение, но тратит на это больше времени.

Заключение.

В ходе работы изучен и реализован алгоритм имитации отжига, а также проведён анализ результатов работы алгоритма и сравнение алгоритма с методом ближайшего соседа. Из полученных данных сделаны выводы:

- Алгоритм имитации отжига может выдавать лучшие решения, чем метод ближайшего соседа, который не всегда выдаёт оптимальные решения;
- Однако сложность алгоритма имитации отжига выше, чем линейная сложность метода ближайшего соседа.

Список литературы.

1. Introducing Python. Modern computing in simple packages / Bill Lobanovic.
2. http://www.machinelearning.ru
3. Ссылка на код https://github.com/ksenkap/annealing