**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**Факультет прикладной математики – процессов управления**

**отчет**

**по лабораторной работе**

**по дисциплине «Алгоритмы и структуры данных»**

**на тему «Реализация метода 2D ССА в приложении к обработке изображений лиц»**

|  |  |  |
| --- | --- | --- |
| Студент гр. 22.Б15 |  | Агишев А.Б. |
| Преподаватель |  | Дик А.Г. |

**Санкт-Петербург**

**2024 г.**

**Содержание**

[1. Цель работы 1](#_Toc151413474)

[2. Задача 1](#_Toc151413475)

[3. Теоретическая часть 1](#_Toc151413476)

[4. Описание алгоритма 1](#_Toc151413477)

[5. Описание программы 3](#_Toc151413478)

[5.1 Описание классов 3](#_Toc151413479)

[5.2. Описание функций 3](#_Toc151413480)

[5.3. Описание переменных 4](#_Toc151413481)

[6. Рекомендации пользователю 6](#_Toc151413482)

[7. Рекомендации программисту 6](#_Toc151413483)

[8. Контрольный пример 7](#_Toc151413484)

[9. Заключение 8](#_Toc151413485)

# **Цель работы**

Реализовать и исследовать алгоритм двухмерного канонического анализа корреляций (2D CCA) для двух наборов данных: каскадного и параллельного.

# **Задача**

Задачи работы включают в себя разработку и реализацию алгоритмов 2D CCA для каскадного и параллельного наборов данных, проведение процедуры РРПП, анализ корреляционных связей методами PLS и CCA, создание программы для взаимной реконструкции объектов, а также оценку производительности и качества алгоритмов. В результате будет выполнен сравнительный анализ результатов и сформулированы выводы о применимости и эффективности методов PLS и CCA.

# **Теоретическая часть**

*Каноническая корреляция (Canonical Correlation)*: Мера связи между двумя множествами переменных. Каноническая корреляция оценивает линейные зависимости между двумя множествами переменных и предоставляет значения корреляции, называемые каноническими корреляциями.

*Канонические переменные (Canonical Variables)*: Линейные комбинации исходных переменных, которые максимизируют каноническую корреляцию между двумя множествами переменных. Каждая каноническая переменная представляет собой линейную комбинацию переменных из каждого множества.

*2D ССА (Two-Dimensional Canonical Correlation Analysis)*: Метод анализа, который обобщает каноническую корреляцию на случай двухмерных данных. В 2D ССА исследуются корреляционные связи между двумя двумерными множествами данных.

*RRPP (Randomized Response Permutation Procedure)*: Метод случайного перемешивания и перестановки, применяемый для оценки статистической значимости корреляционных коэффициентов. RRPP позволяет оценить, насколько наблюдаемая корреляция значима в сравнении с корреляциями, которые могли бы возникнуть случайно.

*PLS (Partial Least Squares)*: Метод регрессии, используемый для моделирования зависимости между наборами переменных. PLS находит линейные комбинации исходных переменных (как входов), которые наилучшим образом объясняют целевые переменные (как выходы).

# **Описание алгоритма**

1. Загрузка изображений из указанной директории. Изображения предварительно обрабатываются для улучшения резкости и добавления гауссовского шума.
2. Обучение моделей 2D CCA (2DCCA Cascade и 2DCCA Parallel).
3. Выполнение процедуры RRPP (Randomized Response Permutation Procedure) для оценки статистической значимости корреляционных коэффициентов.
4. Реконструкция объектов с использованием моделей PLS и CCA.
5. Анализ корреляционных связей в наборах данных с помощью методов PLS и CCA.
6. Вывод результатов работы алгоритма, включая обучение моделей, выполнение RRPP, реконструкцию объектов и анализ корреляционных связей.

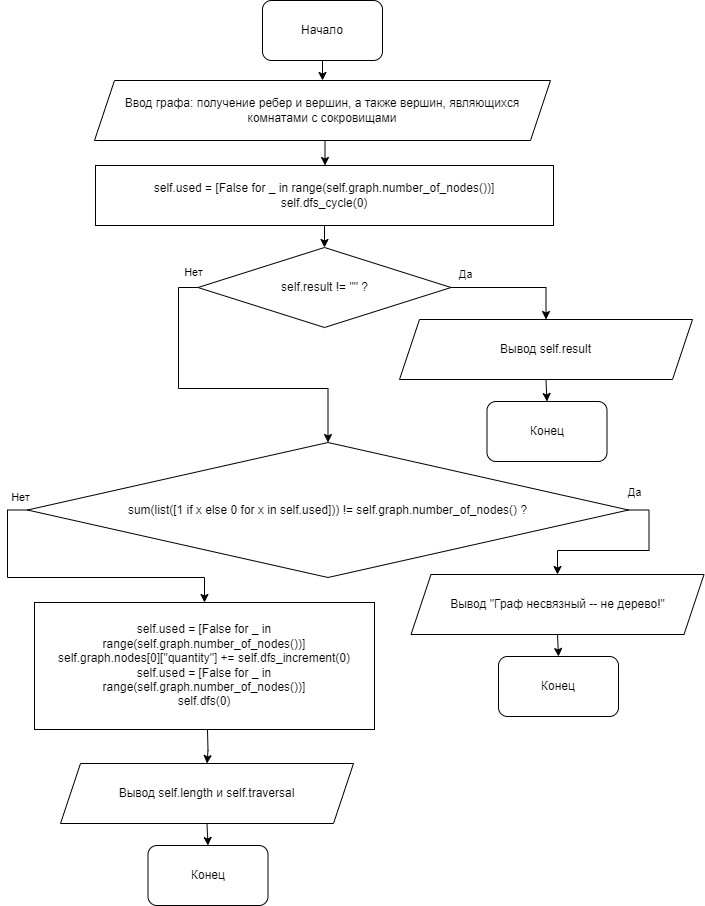


Рисунок 4.1. Блок-схема алгоритма.

# **Описание программы**

## **Описание классов**

В программе используется 3 класса: 2 связанных с интерфейсом программы, 1 связанный с поиском решения задачи. В таблице 5.1 представлено описание классов.

*Таблица 5.1. Описание классов*

|  |  |  |
| --- | --- | --- |
| Имя класса | Наследование | Описание |
| Minimum\_Tree\_Traversal | — | Поиск решения задачи |
| GraphEditor | — | Редактирование графа в интерфейсе |
| Interface | — | Графический пользовательский интерфейс |

## **5.2. Описание функций**

Описание функций класса *«Minimum\_Tree\_Traversal»* представлено в таблице 5.2.

*Таблица 5.2. Описание функций класса «Minimum\_Tree\_Traversal»*

|  |  |  |
| --- | --- | --- |
| Входные данные | Имя | Описание |
| Экземпляр редактора графа | \_\_init\_\_ | Инициализация класса |
| Текущая вершина, предок вершины | dfs\_cycle | Обнаружение циклов в графе |
| Текущая вершина | dfs\_increment | Инкрементация количества сокровищ в каждой вершине |
| Текущая вершина | dfs | Обход графа для поиска минимального пути к сокровищам |
| — | algorithm | Обход графа для поиска минимального пути к сокровищам |

Описание функций класса *«GraphEditor»* представлено в таблице 5.3.

*Таблица 5.3. Описание функций класса «GraphEditor»*

|  |  |  |
| --- | --- | --- |
| Входные данные | Имя | Описание |
| Аргументы | \_\_init\_\_ | Инициализация класса |
| Координаты вершины в интерфейсе | on\_left\_click | Создание вершины в графе |
| Координаты вершины в интерфейсе | on\_middle\_click | Пометка вершины как комнаты с сокровищами |
| Координаты вершины в интерфейсе | on\_right\_click | Создание ребра между вершинами в графе |
| Координаты вершины в интерфейсе | get\_clicked\_vertex | Проверка нажатия на вершину графа |
| — | clear\_graph | Очистка интерфейса графа |

Описание функций класса *«Interface»* представлено в таблице 5.4.

*Таблица 5.4. Описание функций класса «Interface»*

|  |  |  |
| --- | --- | --- |
| Входные данные | Имя | Описание |
| — | \_\_init\_\_ | Инициализация класса |
| — | create\_interface | Создание элементов в интерфейсе |
| Событие | prevent\_typing | Запрет на редактирование  элемента интерфейса |
| — | clear\_output | Очистка интерфейса |
| — | threading\_run | Запуск функции *«start\_process»* в отдельном потоке |
| — | start\_process | Запуск решения поиска задачи о коммивояжере |

## **5.3. Описание переменных**

*Таблица 5.5. Описание переменных функции «algorithm»*

|  |  |  |
| --- | --- | --- |
| Тип | Имя | Описание |
| List[bool] | self.used | Массив с отметкой о посещении вершин |
| List[int] | self.traversal | Обход дерева |
| str | result | Строка с результатом выполнения алгоритма (длина пути и путь) |

Полный код программы представлен в Приложении А.

# **Рекомендации пользователю**

Верхнее поле устанавливает редактор для графа с возможностью задания графа путем добавления вершин и ребер. Левой кнопкой мыши создается вершина в графе, а правая кнопка отвечает за создание ребра между двумя вершинам. Колесиком мыши вершина графа помечается как комната с сокровищем.

Кнопка *«Рассчитать»* отвечает за старт поиска решения задачи. Ответ, а также итоговый обход графа выводится в интерфейс.

Кнопка *«Очистить»* отвечает за очистку интерфейса от всех пользовательских данных.

Для завершения работы нажмите на крестик в левом верхнем углу.

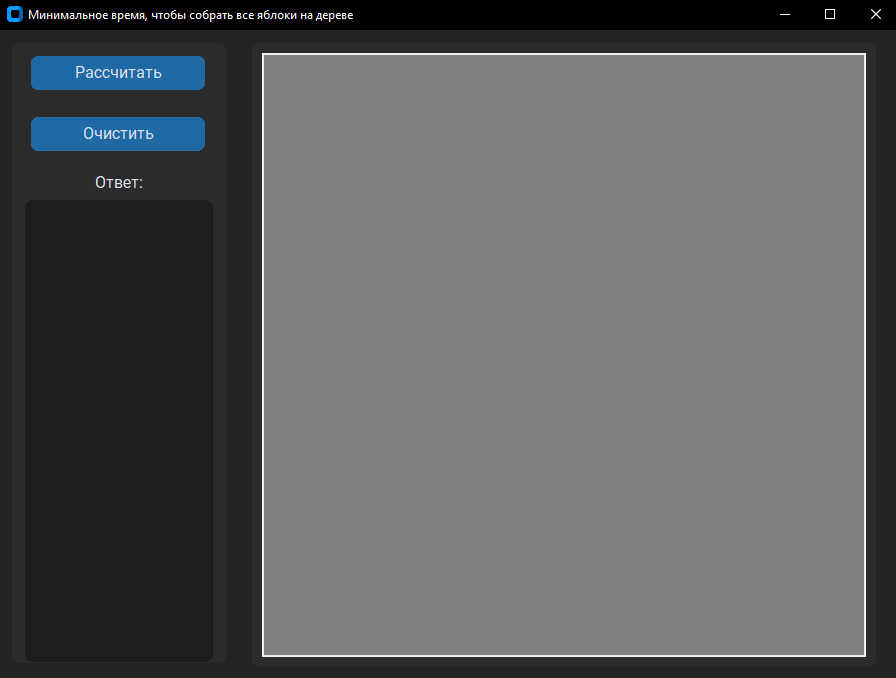
# **Рекомендации программисту**

Для запуска программы необходим Python версии не ниже 3.10.6. Предварительно необходимо установить библиотеки: customtkinter версии не ниже 5.2.0, networkx версии не ниже 3.0.

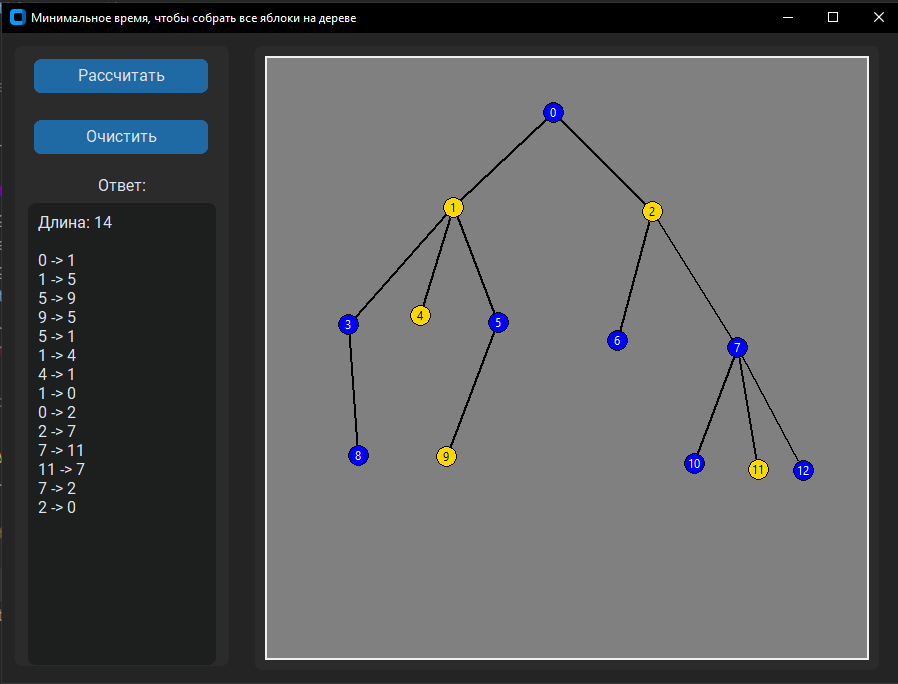
Минимальное необходимое место на диске: 0.5 МБ. Минимальное необходимое количество оперативной памяти: 100 МБ.

# **Контрольный пример**

В данном разделе представлен контрольный пример, демонстрирующий способность программы находить оптимальный порядок обхода дерева.



*Рисунок 8.1. Интерфейс программы*



*Рисунок 8.2. Вывод решения задачи*

# **Заключение**

В ходе работы был разработан алгоритм и программа для нахождения оптимального маршрута в лабиринте катакомб с целью сбора сокровищ и возврата к точке входа. Применение алгоритма глубинного поиска в глубину позволило эффективно исследовать структуру дерева лабиринта, определить оптимальный маршрут и минимизировать время перемещения между комнатами. Полученные результаты могут быть полезны для планирования и оптимизации маршрутов в подобных лабиринтах, а также в других задачах, требующих поиска оптимальных путей в сложных структурах.

1. **Приложение А**

import threading  
import networkx as nx  
import customtkinter as ctk  
  
  
# Function tree algorithm  
class Minimum\_Tree\_Traversal:  
 def \_\_init\_\_(self, graph\_editor):  
 self.graph\_editor = graph\_editor  
 self.edges = graph\_editor.edges  
 self.graph = graph\_editor.graph  
 self.traversal = []  
 self.used = []  
 self.result = ""  
  
 # DFS algo finding cycle  
 def dfs\_cycle(self, v, p=-1):  
 self.used[v] = True  
 for u in self.graph.neighbors(v):  
 if not self.used[u]:  
 self.dfs\_cycle(u, v)  
 elif u != p:  
 self.result = "Граф содержит цикл -- не дерево!"  
 break  
  
 # DFS algo increment quantity  
 def dfs\_increment(self, v):  
 self.used[v] = True  
 for u in self.graph.neighbors(v):  
 if not self.used[u]:  
 self.graph.nodes[v]["quantity"] += self.dfs\_increment(u)  
  
 return self.graph.nodes[v]["quantity"] + 1 if self.graph.nodes[v]["label"] == "Treasure" else self.graph.nodes[v]["quantity"]  
  
 # DFS algo  
 def dfs(self, v):  
 self.used[v] = True  
  
 if self.graph.nodes[v]["quantity"] == 0:  
 if self.graph.nodes[v]["label"] == "Treasure":  
 self.traversal.append(v)  
 return  
  
 self.traversal.append(v)  
 for u in self.graph.neighbors(v):  
 if not self.used[u]:  
 self.dfs(u)  
 if self.traversal[-1] != v:  
 self.traversal.append(v)  
  
 # Algorithm  
 def algorithm(self):  
 self.used = [False for \_ in range(self.graph.number\_of\_nodes())]  
 self.dfs\_cycle(0)  
  
 if self.result != "":  
 return self.result  
  
 if sum(list([1 if x else 0 for x in self.used])) != self.graph.number\_of\_nodes():  
 return "Граф несвязный -- не дерево!"  
  
 self.used = [False for \_ in range(self.graph.number\_of\_nodes())]  
 self.graph.nodes[0]["quantity"] += self.dfs\_increment(0)  
  
 self.used = [False for \_ in range(self.graph.number\_of\_nodes())]  
 self.dfs(0)  
  
 result = f"Длина: {len(self.traversal) - 1}\n\n"  
 for i in range(len(self.traversal) - 1):  
 result += f'{self.traversal[i]} -> {self.traversal[i + 1]}\n'  
 return result  
  
  
# Class for editing a graph on a canvas  
class GraphEditor(ctk.CTkCanvas):  
 def \_\_init\_\_(self, master, \*\*kwargs):  
 super().\_\_init\_\_(master, \*\*kwargs)  
 self.bind("<Button-1>", self.on\_left\_click)  
 self.bind("<Button-3>", self.on\_right\_click)  
 self.bind("<Button-2>", self.on\_middle\_click)  
 self.vertices = []  
 self.edges = []  
 self.selected\_vertex = None  
 self.graph = nx.DiGraph()  
  
 # Function to handle left-click events on the canvas  
 def on\_left\_click(self, event):  
 x, y = event.x, event.y  
 self.create\_oval(x - 10, y - 10, x + 10, y + 10, fill="blue", tags="vertex")  
 self.create\_text(x, y, text=str(len(self.graph)), fill="white", tags="vertex\_text")  
 self.vertices.append((x, y))  
 self.graph.add\_node(len(self.graph), label="Room", quantity=0)  
  
 # Function to handle middle-click events on the canvas (mouse wheel click)  
 def on\_middle\_click(self, event):  
 x, y = event.x, event.y  
 vertex = self.get\_clicked\_vertex(x, y)  
 if vertex is not None:  
 vx, vy = self.vertices[vertex]  
 self.create\_oval(vx - 10, vy - 10, vx + 10, vy + 10, fill="gold", tags="vertex")  
 self.create\_text(vx, vy, text=f"{vertex}", fill="black", tags="vertex\_text")  
 nx.set\_node\_attributes(self.graph, {vertex: {"label": "Treasure", "quantity": 0}})  
  
 # Function to handle right-click events on the canvas  
 def on\_right\_click(self, event):  
 x, y = event.x, event.y  
 vertex = self.get\_clicked\_vertex(x, y)  
 if vertex is not None:  
 if self.selected\_vertex is None:  
 self.selected\_vertex = vertex  
 else:  
 start\_x, start\_y = self.vertices[self.selected\_vertex]  
 end\_x, end\_y = self.vertices[vertex]  
 length = ((end\_x - start\_x) \*\* 2 + (end\_y - start\_y) \*\* 2) \*\* 0.5  
 if length == 0:  
 return  
 arrow\_offset = 10  
 sx = start\_x + (end\_x - start\_x) \* (arrow\_offset / length)  
 sy = start\_y + (end\_y - start\_y) \* (arrow\_offset / length)  
 ex = end\_x - (end\_x - start\_x) \* (arrow\_offset / length)  
 ey = end\_y - (end\_y - start\_y) \* (arrow\_offset / length)  
 self.create\_line(sx, sy, ex, ey, width=2)  
 self.edges.append((self.selected\_vertex, vertex))  
 self.edges.append((vertex, self.selected\_vertex))  
 self.graph.add\_edge(self.selected\_vertex, vertex)  
 self.graph.add\_edge(vertex, self.selected\_vertex)  
 self.selected\_vertex = None  
  
 # Function to get the index of a clicked vertex on the canvas  
 def get\_clicked\_vertex(self, x, y):  
 for i, (vx, vy) in enumerate(self.vertices):  
 if (x - vx) \*\* 2 + (y - vy) \*\* 2 <= 100:  
 return i  
 return None  
  
 # Function to clear the graph on the canvas  
 def clear\_graph(self):  
 self.vertices = []  
 self.edges = []  
 self.graph.clear()  
 self.delete("all")  
  
  
# Class for the graphical user interface  
class Interface(ctk.CTk):  
 def \_\_init\_\_(self):  
 ctk.CTk.\_\_init\_\_(self)  
 ctk.set\_appearance\_mode("dark")  
 ctk.set\_default\_color\_theme("blue")  
 self.title("Минимальное время, чтобы собрать все яблоки на дереве")  
 self.geometry("720x520")  
 self.frame1 = None  
 self.frame2 = None  
 self.clear\_button = None  
 self.answer\_label = None  
 self.graph\_editor = None  
 self.output\_text = None  
 self.process\_button = None  
 self.create\_interface()  
  
 # Function to create the graphical user interface  
 def create\_interface(self):  
 self.frame1 = ctk.CTkFrame(self)  
 self.frame1.grid(row=0, column=0, padx=10, pady=10, sticky="n")  
  
 self.process\_button = ctk.CTkButton(self.frame1, text="Рассчитать", command=self.threading\_run)  
 self.process\_button.pack(side="top", padx=10, pady=10)  
  
 self.clear\_button = ctk.CTkButton(self.frame1, text="Очистить", command=self.clear\_output)  
 self.clear\_button.pack(side="top", padx=10, pady=10)  
  
 self.answer\_label = ctk.CTkLabel(self.frame1, text="Ответ:")  
 self.answer\_label.pack(side="top", padx=10, fill=ctk.BOTH)  
  
 self.output\_text = ctk.CTkTextbox(self.frame1, height=370, width=150)  
 self.output\_text.pack(side="top", padx=10)  
 self.output\_text.bind("<KeyPress>", self.prevent\_typing)  
  
 self.frame2 = ctk.CTkFrame(self)  
 self.frame2.grid(row=0, column=1, padx=10, pady=10, sticky="n")  
  
 self.graph\_editor = GraphEditor(self.frame2, width=600, height=600, bg="grey")  
 self.graph\_editor.pack(side="top", padx=10, pady=10)  
  
 # Function to prevent typing in the output textbox  
 def prevent\_typing(self, event):  
 return "break"  
  
 # Function to clear the output textbox and the graph on the canvas  
 def clear\_output(self):  
 self.graph\_editor.clear\_graph()  
 self.output\_text.delete("1.0", ctk.END)  
  
 # Function to run a process in a separate thread  
 def threading\_run(self):  
 t = threading.Thread(target=self.run\_process)  
 t.start()  
  
 # Function to run a process in the main thread  
 def run\_process(self):  
 self.output\_text.delete("1.0", ctk.END)  
 self.after(10, self.start\_process)  
  
 # Function to start a process  
 def start\_process(self):  
 traversal = Minimum\_Tree\_Traversal(self.graph\_editor)  
 result = traversal.algorithm()  
 self.output\_text.insert(ctk.END, result)  
  
  
# Main block to run the GUI application  
if \_\_name\_\_ == "\_\_main\_\_":  
 gui = Interface()  
 gui.mainloop()

*Листинг программы.*