K-means 聚类项目汇报（中俄对照 | 简单口语版）

> 项目：用 ***K-means*** 做图片颜色聚类（*Color Quantization*），并用 `mrjob` 模拟 ***MapReduce*** 来跑每次迭代。

（说明 / Примечание） - 下面我会按"每个文件"来讲，每个文件从头到尾完整讲完，这样项目里每一块功能代码都有清晰的解释。

---

0) 你可能会卡的发音（中俄对照）/ Произношение (если сложно)

常见读法（中文 → 俄语怎么读） - `K`：K 值 / **ка** - `L1`：L1 / **эль один** - `1.0`：一点零 / **один целых ноль десятых**（口语也常说：**один ноль**） - `300x300`：三百乘三百 / **триста на триста** - `>`：大于 / **больше** - `<`：小于 / **меньше** - `≥`：大于等于 / **больше или равно** - `≤`：小于等于 / **меньше или равно** - `|x|`：绝对值 / **модуль икс** - `--centroids-file`：两个短横线参数 / **два дефиса центроиды файл**（口语：**параметр "centroids file"**）

---

1) 项目目标 / Цель проекта

中文（口语） - 我的项目是用 K-means 把图片的颜色"压缩"。 - 比如原图有很多颜色，我用 K 个中心点（K 种颜色）来代表整张图。 - 最后输出一张新图片，看起来差不多，但是颜色更少。

Русский (разговорно) - Мой проект использует K-means, чтобы "сжать" цвета изображения. - В исходной картинке много цветов, а я оставляю только **K** цветов (K центроидов). - На выходе получается новая картинка: выглядит похоже, но цветов меньше.

---

2) 算法流程 / Алгоритм (общая схема)

中文（一步一步） 1. 读入图片 → 转成像素文本 `pixels_input.txt` 2. 随机选 K 个像素当初始中心 3. 重复迭代： - **Map**：把每个像素分到最近的中心（用 L1 距离） - **Reduce**：对每个簇求平均，得到新中心 4. 收敛后，用最终中心重建图片并保存

Русский (по шагам) 1. Картинка → текстовый файл пикселей `pixels_input.txt` 2. Случайно выбираю K пикселей как начальные центроиды 3. Итерации: - **Map**: назначаю каждый пиксель ближайшему центроиду (L1) - **Reduce**: считаю средний цвет в каждом кластере → новый центроид 4. После сходимости восстанавливаю изображение и сохраняю

---

3) 项目依赖 / Зависимости проекта

文件：`requirements.txt` / **Файл:** `requirements.txt`

完整代码 / Полный код

```
mrjob
numpy
Pillow
```

一句中文：我用 `mrjob` 跑 MapReduce，用 `numpy` 算距离，用 `Pillow` 处理图片。

一句俄语：Я использую `mrjob` для MapReduce, `numpy` для расстояния и `Pillow` для изображений.

---

4) 文件详解：`src/download_sample.py` / Файл: `src/download_sample.py`

> 这个文件用于自动下载测试图片，方便快速测试项目。

完整代码 / **Полный код**

```
import urllib.request
import os

def download_sample():
    # Get project root directory (parent of src)
    project_root = os.path.dirname(os.path.dirname(os.path.abspath(__file__)))
    dataset_dir = os.path.join(project_root, "dataset")

    # Ensure dataset directory exists
    if not os.path.exists(dataset_dir):
        os.makedirs(dataset_dir)

    # Use picsum.photos to get a random 300x300 image
    url = "https://picsum.photos/300/300"
    save_path = os.path.join(dataset_dir, "source_image.jpg")

    print(f"Downloading sample image from {url}...")
    try:
        urllib.request.urlretrieve(url, save_path)
        print(f"Download successful! Image saved to: {save_path}")
        print("You can now run 'python3 main.py' to test.")
    except Exception as e:
        print(f"Download failed: {e}")

if __name__ == "__main__":
    download_sample()
```

功能点逐行讲解 / Объяснение по строкам

功能点 1：定位项目根目录与 dataset 路径

```
project_root = os.path.dirname(os.path.dirname(os.path.abspath(__file__)))
dataset_dir = os.path.join(project_root, "dataset")
```

- 中文：我用当前脚本位置往上找项目根目录，然后拼出 `dataset/` 路径。 - **Русский**：Я беру путь скрипта, поднимаюсь к корню проекта и получаю папку `dataset/`.

功能点 2：如果 dataset 不存在就创建

```
if not os.path.exists(dataset_dir):
    os.makedirs(dataset_dir)
```

- 中文：第一次运行时可能没有 `dataset` 文件夹，我就自动创建。 - **Русский**：Если папки `dataset` нет, я создаю её автоматически.

功能点 3：下载示例图片

```
url = "https://picsum.photos/300/300"
save_path = os.path.join(dataset_dir, "source_image.jpg")
urllib.request.urlretrieve(url, save_path)
```

- 中文：我从网上下载一张 300x300 的随机图片作为测试输入。 - **Русский**：Я скачиваю случайную картинку 300x300 как тестовый вход.

功能点 4：异常处理

```
try:
    urllib.request.urlretrieve(url, save_path)
    print(f"Download successful! Image saved to: {save_path}")
except Exception as e:
    print(f"Download failed: {e}")
```

- 中文：网络有问题也没关系，失败会打印错误，不会直接崩掉。 - **Русский**：Если сеть не работает, будет сообщение об ошибке, и программа не упадёт.

功能点 5：脚本入口

```
if __name__ == "__main__":
    download_sample()
```

- 中文：只有直接运行这个文件时才会下载图片。 - **Русский**：Скачивание запускается только если я запускаю файл напрямую.

---

5) 文件详解：`src/math_utils.py` / Файл: `src/math_utils.py`

> 这个文件定义距离函数，满足作业要求：不用欧氏距离。

完整代码 / Полный код

```
import numpy as np


def manhattan_distance(point1, point2):
    """
    Calculate the Manhattan distance (L1 Norm) between two points.

    Args:
        point1 (list or np.array): Coordinates of the first point (e.g., [R, G, B]).
        point2 (list or np.array): Coordinates of the second point (e.g., [R, G, B]).

    Returns:
        float: The Manhattan distance.
    """
    p1 = np.array(point1)
    p2 = np.array(point2)
    return np.sum(np.abs(p1 - p2))
```

功能点讲解 / Объяснение

功能点：曼哈顿距离（**L1**）

```
def manhattan_distance(point1, point2):
    p1 = np.array(point1)
    p2 = np.array(point2)
    return np.sum(np.abs(p1 - p2))
```

- 中文：我用 L1 距离（曼哈顿距离），不用欧氏距离（L2），满足作业要求。公式是每一维的绝对值差相加。 - **Русский**：Я использую L1 (манхэттенское расстояние), а не L2, чтобы соответствовать требованиям. Формула — сумма модулей разностей.

公式 / **Формула** $$d(\mathbf{x},\mathbf{c}) = \sum_i |x_i - c_i|$$

---

6) 文件详解： `src/image_utils.py` / Файл: `src/image_utils.py`

> 这个文件负责图片与文本之间的转换，以及中心点的读写。

完整代码 / **Полный код**

```
from PIL import Image
import numpy as np
import os
from src.math_utils import manhattan_distance

def image_to_pixels(image_path, output_text_path):
    """
    Reads an image and converts it to a text file where each line is:
    row_id, col_id, R, G, B
    """
    if not os.path.exists(image_path):
        raise FileNotFoundError(f"Image file not found: {image_path}")

    img = Image.open(image_path)
    img = img.convert('RGB')
    width, height = img.size
    pixels = np.array(img)

    with open(output_text_path, 'w') as f:
        for r in range(height):
            for c in range(width):
                R, G, B = pixels[r, c]
                f.write(f"{r},{c},{R},{G},{B}\n")

    print(f"Converted image {image_path} to text {output_text_path}. Size: {width}x{height}")
    return width, height

def reconstruct_image(pixels_path, centroids, width, height, output_image_path):
    """
    Reconstructs the image using the final centroids.
    Each pixel is replaced by the color of its nearest centroid.
    """
    new_pixels = np.zeros((height, width, 3), dtype=np.uint8)
```

```
    print("Reconstructing image...")
    with open(pixels_path, 'r') as f:
        for line in f:
            parts = list(map(int, line.strip().split(',')))
            r_idx, c_idx = parts[0], parts[1]
            pixel_rgb = parts[2:]

            min_dist = float('inf')
            nearest_centroid = None

            for centroid in centroids:
                dist = manhattan_distance(pixel_rgb, centroid)
                if dist < min_dist:
                    min_dist = dist
                    nearest_centroid = centroid

            new_pixels[r_idx, c_idx] = nearest_centroid

    img = Image.fromarray(new_pixels)
    img.save(output_image_path)
    print(f"Saved reconstructed image to {output_image_path}")

def save_centroids(centroids, filepath):
    with open(filepath, 'w') as f:
        for c in centroids:
            f.write(','.join(map(str, c)) + '\n')

def load_centroids(filepath):
    centroids = []
    if not os.path.exists(filepath):
        return []
    with open(filepath, 'r') as f:
        for line in f:
            if line.strip():
```

```
                centroids.append(list(map(float, line.strip().split(','))))
    return centroids
```

功能点逐个讲解 / Объяснение по функциям

功能点 1：`image_to_pixels` - 图片转文本

```
def image_to_pixels(image_path, output_text_path):
    if not os.path.exists(image_path):
        raise FileNotFoundError(f"Image file not found: {image_path}")

    img = Image.open(image_path)
    img = img.convert('RGB')
    width, height = img.size
    pixels = np.array(img)

    with open(output_text_path, 'w') as f:
        for r in range(height):
            for c in range(width):
                R, G, B = pixels[r, c]
                f.write(f"{r},{c},{R},{G},{B}\n")

    return width, height
```

- **中文**： - 先检查输入图片是否存在，不存在就报错。 - 把图片读成 RGB 格式，转成 numpy 数组。 - 每个像素写成一行 `row,col,R,G,B`，这样 MapReduce 可以逐行处理。 - 返回宽高给后续重建使用。 - **Русский**： - Сначала проверяю, есть ли картинка. Если нет — выбрасываю ошибку. - Читаю картинку в RGB и получаю массив пикселей. - Каждый пиксель записываю строкой `row,col,R,G,B` для MapReduce. - Возвращаю ширину и высоту для восстановления.

功能点 2：`reconstruct_image` - 用中心重建图片

```
def reconstruct_image(pixels_path, centroids, width, height, output_image_path):
    new_pixels = np.zeros((height, width, 3), dtype=np.uint8)

    with open(pixels_path, 'r') as f:
        for line in f:
            parts = list(map(int, line.strip().split(',')))
            r_idx, c_idx = parts[0], parts[1]
            pixel_rgb = parts[2:]

            min_dist = float('inf')
            nearest_centroid = None

            for centroid in centroids:
                dist = manhattan_distance(pixel_rgb, centroid)
                if dist < min_dist:
                    min_dist = dist
                    nearest_centroid = centroid

            new_pixels[r_idx, c_idx] = nearest_centroid

    img = Image.fromarray(new_pixels)
    img.save(output_image_path)
```

- **中文**： - 先建一个全黑的空图数组。 - 逐行读像素文本，解析坐标和 RGB。 - 用 L1 距离找到最近的中心，把像素颜色替换成中心颜色。 - 最后把数组转回图片对象并保存。 - **Русский**： - Сначала создаю пустой массив (чёрная картинка). - Читаю текст пикселей построчно, парсю координаты и RGB. - Нахожу ближайший центроид по L1 и заменяю цвет пикселя. - Преобразую массив в изображение и сохраняю.

功能点 3: `save_centroids` - 保存中心

```
def save_centroids(centroids, filepath):
    with open(filepath, 'w') as f:
        for c in centroids:
            f.write(','.join(map(str, c)) + '\n')
```

- 中文：每个中心写成一行 `R,G,B`，下一轮 MRJob 能直接读取。 - **Русский**：Каждый центроид — строка `R,G,B`, следующая итерация MRJob читает напрямую.

功能点 4：`load_centroids` - 读取中心

```
def load_centroids(filepath):
    centroids = []
    if not os.path.exists(filepath):
        return []
    with open(filepath, 'r') as f:
        for line in f:
            if line.strip():
                centroids.append(list(map(float, line.strip().split(','))))
    return centroids
```

- 中文：如果文件不存在就返回空列表，否则逐行读回中心。 - **Русский**：Если файла нет — возвращаю пустой список, иначе читаю центроиды построчно.

---

7) 文件详解：`src/mr_kmeans.py` / Файл: `src/mr_kmeans.py`

> 这个文件是 *MapReduce* 的核心，实现一轮 *K-means* 迭代。

完整代码 / Полный код

```
from mrjob.job import MRJob
from mrjob.step import MRStep
import sys
from src.math_utils import manhattan_distance

class MRKMeans(MRJob):

    def configure_args(self):
        super(MRKMeans, self).configure_args()
        self.add_file_arg('--centroids-file', help='Path to the centroids file')

    def load_centroids(self):
        self.centroids = []
        try:
            with open(self.options.centroids_file, 'r') as f:
                for line in f:
                    if line.strip():
                        parts = line.strip().split(',')
                        self.centroids.append([float(p) for p in parts])
        except Exception as e:
            sys.stderr.write(f"Error loading centroids: {e}\n")

    def mapper_init(self):
        self.load_centroids()

    def mapper(self, _, line):
        try:
            parts = list(map(int, line.strip().split(',')))
            pixel = parts[2:]  # [R, G, B]

            min_dist = float('inf')
            nearest_idx = -1

            for idx, centroid in enumerate(self.centroids):
```

```
                dist = manhattan_distance(pixel, centroid)
                if dist < min_dist:
                    min_dist = dist
                    nearest_idx = idx

            yield nearest_idx, (pixel[0], pixel[1], pixel[2], 1)

        except ValueError:
            pass

    def reducer(self, key, values):
        sum_r, sum_g, sum_b, count = 0, 0, 0, 0

        for r, g, b, c in values:
            sum_r += r
            sum_g += g
            sum_b += b
            count += c

        if count > 0:
            new_r = sum_r / count
            new_g = sum_g / count
            new_b = sum_b / count
            yield key, (new_r, new_g, new_b)

if __name__ == '__main__':
    MRKMeans.run()
```

功能点逐个讲解 / Объяснение по функциям

功能点 1：MRJob 类声明

```
class MRKMeans(MRJob):
    ...
```

- 中文：这个类就是"一轮 K-means"的 MapReduce 任务。 - **Русский**：Этот класс — MapReduce-задача для одной итерации K-means.

功能点 2：配置命令行参数

```
def configure_args(self):
    super(MRKMeans, self).configure_args()
    self.add_file_arg('--centroids-file', help='Path to the centroids file')
```

- 中文：添加 `--centroids-file` 参数，让主程序可以把中心文件传进来。 - **Русский**：Добавляю параметр `--centroids-file`, чтобы главная программа передала файл центроидов.

功能点 3：加载中心文件

```
def load_centroids(self):
    self.centroids = []
    try:
        with open(self.options.centroids_file, 'r') as f:
            for line in f:
                if line.strip():
                    parts = line.strip().split(',')
                    self.centroids.append([float(p) for p in parts])
    except Exception as e:
        sys.stderr.write(f"Error loading centroids: {e}\n")
```

- 中文：中心文件每行是 `R,G,B`，我读出来变成浮点数列表。如果读取失败，错误输出到 stderr。 - **Русский**：В файле каждая строка `R,G,B`, я читаю и делаю список float. Ошибки пишу в stderr.

功能点 4：Mapper 初始化

```
def mapper_init(self):
    self.load_centroids()
```

- 中文：mapper 处理数据前先读中心文件，这样每行像素都能用同一组中心。 - **Русский**：Перед обработкой строк mapper загружает центроиды из файла.

功能点 5：Mapper - 给每个像素分配最近中心

```
def mapper(self, _, line):
    try:
        parts = list(map(int, line.strip().split(',')))
        pixel = parts[2:]  # [R, G, B]

        min_dist = float('inf')
        nearest_idx = -1

        for idx, centroid in enumerate(self.centroids):
            dist = manhattan_distance(pixel, centroid)
            if dist < min_dist:
                min_dist = dist
                nearest_idx = idx

        yield nearest_idx, (pixel[0], pixel[1], pixel[2], 1)

    except ValueError:
        pass
```

- 中文： - 解析输入行：前两位是坐标，后三位是 RGB。 - 用 L1 距离找到最近中心。 - 输出 `(簇编号, (R, G, B, 1))`，其中 1 用于后面计算平均。 - 如果某行格式不对，直接跳过。 - **Русский**： - Парсю строку: первые два числа — координаты, последние три — RGB. - Нахожу ближайший центроид по L1. - Выдаю `(номер кластера, (R, G, B, 1))`, где 1 — для подсчёта среднего. - Плохие строки пропускаю.

功能点 6：Reducer - 计算新中心

```
def reducer(self, key, values):
    sum_r, sum_g, sum_b, count = 0, 0, 0, 0

    for r, g, b, c in values:
        sum_r += r
        sum_g += g
        sum_b += b
        count += c

    if count > 0:
        new_r = sum_r / count
        new_g = sum_g / count
        new_b = sum_b / count
        yield key, (new_r, new_g, new_b)
```

- **中文**： - reducer 把同一簇的所有像素颜色相加并计数。 - 做平均得到新的中心颜色。 - 只有 count > 0 才输出，防止除以 0（空簇不输出）。 - **Русский**： - Reducer суммирует все цвета в кластере и считает количество. - Вычисляю среднее — это новый центроид. - Вывод только если count > 0, чтобы не делить на ноль.

功能点 7：MRJob 运行入口

```
if __name__ == '__main__':
    MRKMeans.run()
```

- **中文**：这样这个文件可以单独作为 MRJob 来跑。 - **Русский**：Так файл можно запускать отдельно как MRJob.

---

8) 文件详解： `main.py` / Файл: `main.py`

> 这是主程序，把所有模块串起来，完成整个 *K-means* 流程。

完整代码 / **Полный код**

```
import os
import sys
import random
import shutil
from src.image_utils import image_to_pixels, reconstruct_image, save_centroids, load_centroid
from src.mr_kmeans import MRKMeans
from src.math_utils import manhattan_distance

# Configuration
PROJECT_ROOT = os.path.dirname(os.path.abspath(__file__))
DATASET_DIR = os.path.join(PROJECT_ROOT, 'dataset')
SRC_DIR = os.path.join(PROJECT_ROOT, 'src')

INPUT_IMAGE = os.path.join(DATASET_DIR, 'source_image.jpg')
PIXELS_FILE = os.path.join(DATASET_DIR, 'pixels_input.txt')
CENTROIDS_FILE = os.path.join(DATASET_DIR, 'initial_centroids.txt')
OUTPUT_IMAGE = os.path.join(DATASET_DIR, 'output_images', 'result.jpg')

K = 5  # Number of clusters
MAX_ITER = 10
THRESHOLD = 1.0  # Convergence threshold

def initialize_centroids(pixels_file, k, output_file):
    print("Initializing centroids...")
    pixels = []
    with open(pixels_file, 'r') as f:
        for line in f:
            parts = list(map(int, line.strip().split(',')))
            pixels.append(parts[2:])  # [R, G, B]

    centroids = random.sample(pixels, k)
    save_centroids(centroids, output_file)
    print(f"Initialized {k} centroids.")
    return centroids
```

```
def run_mr_job(pixels_file, centroids_file):
    args = [pixels_file, '--centroids-file', centroids_file]
    job = MRKMeans(args=args)

    new_centroids_dict = {}

    with job.make_runner() as runner:
        runner.run()
        for key, value in job.parse_output(runner.cat_output()):
            new_centroids_dict[key] = value

    return new_centroids_dict

def main():
    # 1. Preprocessing
    if not os.path.exists(INPUT_IMAGE):
        print(f"Error: {INPUT_IMAGE} not found. Please place an image there.")
        return

    print("Step 1: Converting image to text...")
    width, height = image_to_pixels(INPUT_IMAGE, PIXELS_FILE)

    # 2. Initialization
    print("Step 2: Initializing centroids...")
    current_centroids = initialize_centroids(PIXELS_FILE, K, CENTROIDS_FILE)

    # 3. Iteration
    print("Step 3: Starting K-means iteration...")
    iteration = 0
    while iteration < MAX_ITER:
        print(f"--- Iteration {iteration + 1} ---")

        new_centroids_map = run_mr_job(PIXELS_FILE, CENTROIDS_FILE)
```

```
        new_centroids = []
        max_shift = 0.0

        for i in range(K):
            if i in new_centroids_map:
                new_c = new_centroids_map[i]
                old_c = current_centroids[i]
                shift = manhattan_distance(new_c, old_c)
                if shift > max_shift:
                    max_shift = shift
                new_centroids.append(new_c)
            else:
                print(f"Warning: Cluster {i} is empty. Keeping old centroid.")
                new_centroids.append(current_centroids[i])

        print(f"Max shift: {max_shift}")

        save_centroids(new_centroids, CENTROIDS_FILE)
        current_centroids = new_centroids

        if max_shift < THRESHOLD:
            print("Converged!")
            break

        iteration += 1

    # 4. Post-processing
    print("Step 4: Reconstructing image...")
    output_dir = os.path.dirname(OUTPUT_IMAGE)
    if not os.path.exists(output_dir):
        os.makedirs(output_dir)

    reconstruct_image(PIXELS_FILE, current_centroids, width, height, OUTPUT_IMAGE)
```

```
    print("Done!")


if __name__ == '__main__':
    main()
```

功能点逐个讲解 / Объяснение по функциям

功能点 1：导入模块

```
from src.image_utils import image_to_pixels, reconstruct_image, save_centroids, load_centroid
from src.mr_kmeans import MRKMeans
from src.math_utils import manhattan_distance
```

- 中文：主程序不自己做细节，它调用工具模块来完成每一步。 - **Русский**：Главный файл не делает всё сам, он вызывает утилиты по шагам.

功能点 2：配置路径和超参数

```
INPUT_IMAGE = os.path.join(DATASET_DIR, 'source_image.jpg')
PIXELS_FILE = os.path.join(DATASET_DIR, 'pixels_input.txt')
CENTROIDS_FILE = os.path.join(DATASET_DIR, 'initial_centroids.txt')
OUTPUT_IMAGE = os.path.join(DATASET_DIR, 'output_images', 'result.jpg')

K = 5
MAX_ITER = 10
THRESHOLD = 1.0
```

- 中文：这段设置输入输出路径和 K（聚类数）、迭代次数、收敛阈值。 - **Русский**：Тут задаются пути и параметры: K (число кластеров), число итераций, порог сходимости.

功能点 3：`initialize_centroids` - 随机初始化中心

```
def initialize_centroids(pixels_file, k, output_file):
    pixels = []
    with open(pixels_file, 'r') as f:
        for line in f:
            parts = list(map(int, line.strip().split(',')))
            pixels.append(parts[2:])  # [R, G, B]

    centroids = random.sample(pixels, k)
    save_centroids(centroids, output_file)
    return centroids
```

- 中文：从像素文本里读取所有 RGB，随机抽 K 个颜色当初始中心。 - **Русский**：Читаю все RGB из текста и случайно выбираю K цветов как начальные центроиды.

功能点 4：`run_mr_job` - 运行一轮 MapReduce

```
def run_mr_job(pixels_file, centroids_file):
    args = [pixels_file, '--centroids-file', centroids_file]
    job = MRKMeans(args=args)

    new_centroids_dict = {}

    with job.make_runner() as runner:
        runner.run()
        for key, value in job.parse_output(runner.cat_output()):
            new_centroids_dict[key] = value

    return new_centroids_dict
```

- 中文：把中心文件传给 MRJob，运行后拿到 reducer 输出的新中心（字典形式：key 是簇编号，value 是 RGB）。 - **Русский**：Передаю файл центроидов в MRJob и получаю новые центроиды из reducer (словарь: ключ — индекс кластера, значение — RGB).

功能点 5：`main` - 主流程

步骤 1：检查输入图片并转换

```
if not os.path.exists(INPUT_IMAGE):
    print(f"Error: {INPUT_IMAGE} not found. Please place an image there.")
    return

width, height = image_to_pixels(INPUT_IMAGE, PIXELS_FILE)
```

- 中文：没有输入图片就直接提示并退出；有的话就转成文本。 - **Русский**：Если картинки нет, печатаю сообщение и выхожу; если есть — преобразую в текст.

步骤 2：初始化中心

```
current_centroids = initialize_centroids(PIXELS_FILE, K, CENTROIDS_FILE)
```

- 中文：随机选 K 个像素颜色作为初始中心。 - **Русский**：Случайно выбираю K цветов как начальные центроиды.

步骤 3：迭代循环

```
while iteration < MAX_ITER:
    new_centroids_map = run_mr_job(PIXELS_FILE, CENTROIDS_FILE)

    new_centroids = []
    max_shift = 0.0

    for i in range(K):
        if i in new_centroids_map:
            new_c = new_centroids_map[i]
            old_c = current_centroids[i]
            shift = manhattan_distance(new_c, old_c)
            if shift > max_shift:
                max_shift = shift
            new_centroids.append(new_c)
        else:
            print(f"Warning: Cluster {i} is empty. Keeping old centroid.")
            new_centroids.append(current_centroids[i])

    save_centroids(new_centroids, CENTROIDS_FILE)
    current_centroids = new_centroids

    if max_shift < THRESHOLD:
        print("Converged!")
        break
```

- **中文**： - 每轮运行 MapReduce，拿到新中心。 - 计算每个中心的移动距离，记录最大值。 - 如果某个簇为空（没有像素分到），保留旧中心。 - 如果最大移动距离小于阈值，说明收敛了，停止迭代。 - 否则更新中心文件，继续下一轮。 - **Русский**： - Каждую итерацию запускаю MapReduce и получаю новые центроиды. - Считаю сдвиг каждого центроида, запоминаю максимум. - Если кластер пустой, оставляю старый центроид. - Если максимальный сдвиг меньше порога — сходимость, останавливаюсь. - Иначе обновляю файл центроидов и продолжаю.

步骤 4：重建输出图片

```
output_dir = os.path.dirname(OUTPUT_IMAGE)
if not os.path.exists(output_dir):
    os.makedirs(output_dir)

reconstruct_image(PIXELS_FILE, current_centroids, width, height, OUTPUT_IMAGE)
```

- 中文：确保输出目录存在，用最终中心重建并保存结果图片。 - **Русский**：Создаю папку для результата (если нет), восстанавливаю и сохраняю итоговую картинку.

功能点 6：主程序入口

```
if __name__ == '__main__':
    main()
```

- 中文：保证这个文件作为脚本运行时才执行主流程。 - **Русский**：Главный процесс запускается только при прямом запуске файла.

---

9) 作业要求对照 / Требования

9.1 不用欧几里得距离 / Не использовать евклидову метрику

中文：我用的是曼哈顿距离 L1（`src/math_utils.py`）。

Русский：Я использую манхэттенское расстояние L1 (`src/math_utils.py`).

9.2 至少 3 维 / Не меньше 3 измерений

中文：每个像素是 `[R,G,B]`，所以是 3 维。

Русский：Каждый пиксель — `[R,G,B]`, значит 3 измерения.

9.3（加分）用图片 / (Доп.) Использовать изображения

中文：我输入是图片，数据来自真实像素，不是人工点坐标。

Русский: Вход — изображение, данные — реальные пиксели, не точки.

---

10) 怎么运行 / Как запустить

中文 1. 准备图片：放到 `dataset/source_image.jpg` 2. 安装依赖：

```
pip install -r requirements.txt
```

3. 运行：

```
python3 main.py
```

4. 输出：`dataset/output_images/result.jpg`

Русский 1. Подготовить картинку: `dataset/source_image.jpg` 2. Установить зависимости:

```
pip install -r requirements.txt
```

3. Запуск:

```
python3 main.py
```

4. Результат: `dataset/output_images/result.jpg`

---

11) 口语汇报稿（1~2 分钟）/ Устный текст (1–2 минуты)

中文 老师您好，我介绍一下我的作业。 我的项目是用 K-means 做图片的颜色聚类。输入是一张图片，我把它转成像素数据，每个像素是 `[R,G,B]` 三维向量。 作业要求不能用欧氏距离，所以我用的是曼哈顿距离，也就是每一维的绝对值差相加。 然后我用 MapReduce 的思想做 K-means：Mapper 负责把每个像素分配到最近的中心，Reducer

负责对每个簇求平均，得到新的中心。 我在主程序里循环迭代，如果中心点变化小于阈值就停止。最后用最终中心把图片重建并保存。 谢谢老师。

Русский Здравствуйте, я коротко расскажу про свою работу. Мой проект использует K-means для кластеризации цветов изображения. Вход — картинка, я преобразую её в данные пикселей, и каждый пиксель — это 3-мерный вектор `[R,G,B]`. По требованию я не использую евклидову метрику, поэтому применяю манхэттенское расстояние (сумма модулей разностей по координатам). Дальше я реализую K-means в стиле MapReduce: Mapper назначает пиксель ближайшему центроиду, Reducer считает средний цвет внутри кластера и обновляет центроид. В `main.py` я повторяю итерации и останавливаюсь, когда сдвиг центроидов меньше порога. В конце я восстанавливаю изображение и сохраняю результат. Спасибо.

---

12) 老师可能会问的问题 / Возможные вопросы

Q1：为什么用 **L1**，不用 **L2**？ **/ Почему L1, а не L2?** - 中文：因为作业要求不能用欧氏距离，我选 L1（曼哈顿）。它也更简单，算得快。 - **Русский**: Потому что по условию нельзя L2. Я выбрал L1: проще и быстрее считать.

Q2：为什么中心用"平均值"？ **/ Почему центроид = среднее?** - 中文：我这里实现的是经典 K-means 的更新方式（求平均颜色）。 - **Русский**: Я использую стандартное обновление K-means: средний цвет.

Q3：空簇怎么办？ **/ Что если кластер пустой?** - 中文：我保留旧中心，不让它消失，这样程序稳定。 - **Русский**: Я оставляю старый центроид, чтобы алгоритм был стабильным.