# 数据结构说明报告

第二组 任泽高 安轶东 董佳鑫

[数据结构说明报告](#数据结构说明报告)  
 [用户数据结构分析](#用户数据结构分析)  
 [1. 用户数据存储](#X30017e53d4386d8757c1dc5f0b7e20a45e2296e)  
 [2. 数据结构描述](#X987587d84f8290a4b7ccd97ed4129d20d2fb065)  
 [顶层结构](#顶层结构)  
 [内层结构](#内层结构)  
 [3. 字段](#X43cf389602efc6d32e3b8ae6061f7ca7c9f9a76)  
 [4. 功能](#X06e0b20c6b3a7750663d6284c1d730494df9c71)  
 [5. 数据量描述](#X7673fb78e4e2e2176b6bb0fd081b507aeebb9f0)  
 [数据量示例](#数据量示例-1)  
 [地图信息数据结构分析](#地图信息数据结构分析)  
 [1. 结构体分析](#X0a4a8abef3a32c0868072abf3b543d562cd7519)  
 [1.1 Foods 结构体](#X141f75695bea822e5fec623d0f0b6958a5e537c)  
 [1.2 ArcNode 结构体](#Xdc640330d7f14fbef5ff047cf6a5c44cf2f88b7)  
 [1.3 VNode 结构体](#X5ecd8d766337a25b0d7c3693ea13ba35fc69bb6)  
 [1.4 ALGraph 结构体](#Xb35300c7b5aa6d34ce32e0e79097a472577e110)  
 [1.5 Attraction 结构体](#X879f82da24196c998f0292255920fa9de73cb00)  
 [2. JSON 文件分析](#X93cf6faccd925799073af8dfa61a81fe79be58f)  
 [3. JSON 字段描述](#Xf383095407323bad5ade3fe61e2f57734da451d)  
 [1. **Attraction**](#X2c19f587c3a12c9a1d209c996bf65ebdc23e6fa)  
 [2. **builds**](#Xb85f21f91d84bc4c022c08c7936aaabb21b8dcc)  
 [3. **foods**](#X8f87754c9c2c2b853af145eb66790c418add1fc)  
 [4. **routes**](#X4de65c27fb27938366b629c539ac1e1825b3320)  
 [4. 功能分析](#Xba2b4825b74b4f1db704e2c1509321f4a5935cf)  
 [4.1 全局变量](#X2d1b7d2e9107a3eb35b3b70321365c5779c88cc)  
 [4.2 功能函数](#X8881babd27687393a166550f88dff09a3dfe44e)  
 [4.3 数据结构功能和关系](#X0a0e58e1b56c34c47e22b11e45721b59ad7f053)  
 [4.4 数据结构之间的具体关系](#X61fd4b936d628c5da50abe6d8dadbc2810470a0)  
 [4.5 具体代码实现中的数据结构分析](#X5c413119b4662253764abcc7e9ee58472a58ff8)  
 [5. 数据量描述](#X7344c46a63215565a3fea01b7986525c871ef8c)  
 [数据量示例](#数据量示例-2)  
 [日记数据结构分析](#日记数据结构分析)  
 [1. Diary 结构体定义](#Xbe7bef0f00f654f6e65aea18e101a2cd4415a5f)  
 [字段解析](#字段解析)  
 [2. diaries.json 文件格式](#X79ad5e0a014625ccd977cca6ae908ec75155031)  
 [3. 压缩和解压缩机制](#Xdae01765dcb58186ac2a681d0c3f867cb963bb3)  
 [4. 日记数据结构在代码中的使用](#X74ccd8d209082762d532fc37b5de5de1e153416)  
 [5. diaryManagement 类的解析函数](#Xcbbacb4778932ec1eb95894fc3cfc5a80f667e8)  
 [6. 数据量描述](#X5d570477271c3cb3f3820edbf3b8de2bee64ce9)  
 [数据量示例](#数据量示例-3)  
 [路线规划与推荐功能的数据结构分析](#路线规划与推荐功能的数据结构分析)  
 [1. 路线规划相关结构体分析](#X4129d90a47fa73f9e758c42948dd18744a4c767)  
 [1.1 Coordinate 结构体](#X5f3a0d50d0bbf2a8874f5986133644abf8b171e)  
 [1.2 全局变量](#X23d430856065faaecd80d065997feef7252f7d9)  
 [2. 推荐功能相关结构体函数分析](#X8f0cc9c483d8a0187a0d25a766895505753b44b)  
 [2.1 destinationRecommend 结构体](#X3c344a2a2e0033c298a5d828dd6faf83b44caef)  
   
 [3. 数据结构关系](#X2c0ffa0a916b99f82a98fa649c1886e3115caf1)  
 [3.1 Coordinate 与 ArcNode](#X33d1c7ca3896aa55e31a77e76ed5598e3741051)  
 [3.2 ALGraph 与 VNode 和 ArcNode](#X067a381dd9fc63b3e36870bf988450b2247430a)  
 [3.3 destinationRecommend 与 Attraction](#X1200cdb2ed50e8920fb62d2a6661479b9642a1a)  
 [4. 数据量描述](#Xc16313a1d55ff6947addc0c6d12874677a52efd)

## 用户数据结构分析

用户数据结构主要包含一个accounts对象，其中键为用户账号，值为用户密码。通过读取和修改user\_accounts.json文件，实现了用户账号的创建、验证和存在性检查。文件结构设计简单，便于扩展，能够容纳更多用户信息。

### 1. 用户数据存储

用户信息通过JSON格式存储，结构如下：

{
  
 "accounts": {
  
 "1": "1",
  
 "11": "11",
  
 "username1": "password1",
  
 }
  
}

### 2. 数据结构描述

#### 顶层结构

* **accounts**: 顶层字段，存储所有用户账号和密码的集合。

#### 内层结构

* **"1": "1"** 等键值对：每个键值对代表一个用户的账号和密码，其中：
  + **键（key）**：用户账号的字符串表示。
  + **值（value）**：用户密码的字符串表示。

### 3. 字段

1. **accounts 字段**
   * **用途**：存储所有用户的账号和密码信息。
   * **类型**：对象（Object），键为用户账号，值为用户密码。
2. **用户账号（Key）**
   * **用途**：唯一标识一个用户。
   * **类型**：字符串（String），例如 "1", "11", "ren", "123" 等。
   * **特性**：每个账号必须唯一，不同用户不能有相同的账号。
3. **用户密码（Value）**
   * **用途**：用于用户登录时的身份验证。
   * **类型**：字符串（String），例如 "1", "11", "ren", "123" 等。
   * **特性**：每个密码对应一个账号，密码可以是任何字符串。

### 4. 功能

1. **存储用户账号和密码**
   * 将所有用户的账号和密码以键值对的形式存储在 accounts 对象中。
2. **验证用户登录**
   * 根据输入的账号和密码，检查 accounts 对象中是否存在对应的键值对。
   * 如果账号存在且密码匹配，则允许用户登录，否则拒绝。
3. **创建新用户**
   * 在 accounts 对象中添加新的键值对，将新用户的账号和密码存储起来。
4. **检测用户是否存在**
   * 通过检查 accounts 对象中的键来判断某个用户账号是否已经存在。

### 5. 数据量描述

在本项目中，user\_accounts.json 文件存储了多个用户的账号和密码信息。每个账号和密码为字符串类型，通常账号为用户名或ID，密码为用户设置的密码。文件结构允许继续添加更多用户信息。

#### 数据量示例

假设每个账号和密码平均长度为10个字符，20个用户的数据总大小约为400个字符（400B）。随着用户数量的增加，文件大小会线性增长。

## 地图信息数据结构分析

### 1. 结构体分析

#### 1.1 Foods 结构体

class Foods {  
public:  
 string name; // 美食名称  
 int restaurantID; // 餐厅ID  
 string cuisine; // 菜系  
 int popularity; // 受欢迎程度  
};

**功能**：用于存储景区内的美食信息，包括美食名称、餐厅ID、菜系和美食的受欢迎程度。

**字段**：

* name：美食的名称，类型为 string，例如 "烤鸭", "炒鸡肉"。
* restaurantID：美食所在餐厅的ID，类型为 int，例如 1, 2。
* cuisine：美食的菜系，类型为 string，例如 "京菜", "川菜"。
* popularity：美食的受欢迎程度，类型为 int，例如 65, 97。

#### 1.2 ArcNode 结构体

class ArcNode {  
public:  
 int ID; // 边的终点ID  
 string name; // 边的终点名称  
 ArcNode \*next; // 指向下一条边  
 int length; // 边的长度  
 int midx, midy; // 中间点的坐标  
 double walkCongestion; // 步行拥挤度  
 double bikeCongestion; // 骑行拥挤度  
 double electricCongestion; // 电动车拥挤度  
};

**功能**：表示边表结点，代表两个景点之间的道路，包括边的终点ID、名称、指向下一个边的指针、道路长度和各种拥挤度信息。

**字段**：

* ID：边的终点的ID，类型为 int。
* name：边的终点的名字，类型为 string。
* next：指向从起点伸出的下一条边，类型为 ArcNode\*。
* length：边的长度，类型为 int。
* midx：中点的横坐标，类型为 int。
* midy：中点的纵坐标，类型为 int。
* walkCongestion：边的步行拥挤度，类型为 double。
* bikeCongestion：边的骑行拥挤度，类型为 double。
* electricCongestion：边的电动车拥挤度，类型为 double。

#### 1.3 VNode 结构体

class VNode {  
public:  
 string name; // 景点名称  
 bool isFacility; // 是否为设施  
 int x, y; // 景点的坐标  
 ArcNode \*first; // 指向第一条边  
};

**功能**：表示顶点表结点，代表一个景点，包括景点名称、是否为设施、坐标和指向第一条边的指针。

**字段**：

* name：景点的名字，类型为 string。
* isFacility：是否为设施，类型为 bool。
* x：景点的横坐标，类型为 int。
* y：景点的纵坐标，类型为 int。
* first：指向第一条以该景点为起点的边，类型为 ArcNode\*。

#### 1.4 ALGraph 结构体

class ALGraph {  
public:  
 VNode vertexes[MAX\_VERTEX\_NUM]; // 顶点表  
};

**功能**：表示景区的内部地图，使用邻接表表示。

**字段**：

* vertexes：顶点表，表示一个景区内的所有景点，类型为 VNode[MAX\_VERTEX\_NUM]。

#### 1.5 Attraction 结构体

class Attraction {  
public:  
 string name; // 景区名称  
 bool isSchool; // 是否为学校  
 int popularity; // 景区热度  
 double rating; // 景区评分  
 ALGraph graph; // 景区内部地图  
 Foods foods[MAX\_FOOD\_NUM]; // 景区内的美食  
};

**功能**：表示一个景区，包括景区名称、是否为学校、热度、评分、景区内的地图（邻接表）和景区内的美食信息。

**字段**：

* name：景区的名字，类型为 string。
* isSchool：是否为学校，类型为 bool。
* popularity：景区的热度，类型为 int。
* rating：景区的评分，类型为 double。
* graph：景区的内部地图，类型为 ALGraph。
* foods：景区内的美食信息，类型为 Foods[MAX\_FOOD\_NUM]。

### 2. JSON 文件分析

以下是一个示例 JSON 文件，包含景区和校园的信息，包括建筑物、设施、美食和路线等数据。

[  
 {  
 "name": "赵 山",  
 "ID": 0,  
 "isSchool": false,  
 "popular": 2,  
 "rating": 1.72,  
 "builds": [  
 {  
 "name": "李 Cafeteria",  
 "ID": 0,  
 "isFacility": false,  
 "coordinates": [  
 237,  
 137  
 ]  
 },  
 {  
 "name": "赵 食堂",  
 "ID": 1,  
 "isFacility": false,  
 "coordinates": [  
 140,  
 107  
 ]  
 },  
 {  
 "name": "朱 Supermarket",  
 "ID": 2,  
 "isFacility": true,  
 "coordinates": [  
 179,  
 229  
 ]  
 },  
 {  
 "name": "John 游泳池",  
 "ID": 3,  
 "isFacility": true,  
 "coordinates": [  
 267,  
 8  
 ]  
 }  
 ],  
 "foods": [  
 {  
 "name": "烤鸭",  
 "restaurantID": 1,  
 "cuisine": "京菜",  
 "popularity": 65  
 },  
 {  
 "name": "炒鸡肉",  
 "restaurantID": 1,  
 "cuisine": "川菜",  
 "popularity": 97  
 }  
 ],  
 "routes": [  
 {  
 "start": 2,  
 "end": 0,  
 "start\_coordinates": [  
 179,  
 229  
 ],  
 "end\_coordinates": [  
 237,  
 137  
 ],  
 "midpoint": [  
 179,  
 137  
 ],  
 "distance": 150,  
 "walkCongestion": 0.89,  
 "bikeCongestion": 0.9,  
 "electricCongestion": 0.96  
 },  
 {  
 "start": 2,  
 "end": 1,  
 "start\_coordinates": [  
 179,  
 229  
 ],  
 "end\_coordinates": [  
 140,  
 107  
 ],  
 "midpoint": [  
 179,  
 107  
 ],  
 "distance": 161,  
 "walkCongestion": 0.72,  
 "bikeCongestion": -1,  
 "electricCongestion": 0.82  
 }  
 ]  
 },  
 {  
 "name": "Mia University of Technology",  
 "ID": 1,  
 "isSchool": true,  
 "popular": 76,  
 "rating": 2.55,  
 "builds": [  
 {  
 "name": "孙 食堂",  
 "ID": 0,  
 "isFacility": false,  
 "coordinates": [  
 205,  
 110  
 ]  
 },  
 {  
 "name": "何 Cafeteria",  
 "ID": 1,  
 "isFacility": false,  
 "coordinates": [  
 127,  
 198  
 ]  
 },  
 {  
 "name": "John Basketball Court",  
 "ID": 2,  
 "isFacility": true,  
 "coordinates": [  
 235,  
 50  
 ]  
 },  
 {  
 "name": "Lincon 游泳池",  
 "ID": 3,  
 "isFacility": true,  
 "coordinates": [  
 86,  
 237  
 ]  
 }  
 ],  
 "foods": [  
 {  
 "name": "炒鸡腿",  
 "restaurantID": 0,  
 "cuisine": "美国菜",  
 "popularity": 74  
 },  
 {  
 "name": "炒羊腿",  
 "restaurantID": 0,  
 "cuisine": "新疆菜",  
 "popularity": 72  
 }  
 ],  
 "routes": [  
 {  
 "start": 3,  
 "end": 0,  
 "start\_coordinates": [  
 86,  
 237  
 ],  
 "end\_coordinates": [  
 205,  
 110  
 ],  
 "midpoint": [  
 205,  
 237  
 ],  
 "distance": 246,  
 "walkCongestion": 0.69,  
 "bikeCongestion": 0.84,  
 "electricCongestion": 0.67  
 },  
 {  
 "start": 3,  
 "end": 2,  
 "start\_coordinates": [  
 86,  
 237  
 ],  
 "end\_coordinates": [  
 235,  
 50  
 ],  
 "midpoint": [  
 235,  
 237  
 ],  
 "distance": 336,  
 "walkCongestion": 0.64,  
 "bikeCongestion": 0.65,  
 "electricCongestion": 0.54  
 }  
 ]  
 }  
]

### 3. JSON 字段描述

#### 1. **Attraction**

* **name**
  + **用途**：景区的名称。
  + **类型**：字符串（String），例如 "赵 山", "Mia University of Technology"。
* **ID**
  + **用途**：景区的ID。
  + **类型**：整数（Integer），例如 0, 1。
* **isSchool**
  + **用途**：标识是否为学校。
  + **类型**：布尔值（Boolean），例如 true, false。
* **popular**
  + **用途**：景区的热度。
  + **类型**：整数（Integer），例如 2, 76。
* **rating**
  + **用途**：景区的评分。
  + **类型**：双精度浮点数（Double），例如 1.72, 2.55。
* **builds**
  + **用途**：景区内的建筑物信息。
  + **类型**：对象数组。

#### 2. **builds**

* **name**
  + **用途**：建筑物的名称。
  + **类型**：字符串（String），例如 "李 Cafeteria", "赵 食堂"。
* **ID**
  + **用途**：建筑物的ID。
* **类型**：整数（Integer），例如 0, 1。
* **isFacility**
  + **用途**：标识是否为设施。
  + **类型**：布尔值（Boolean），例如 true, false。
* **coordinates**
  + **用途**：建筑物的坐标。
  + **类型**：整数数组（Array of Integers），例如 [237, 137], [140, 107]。

#### 3. **foods**

* **name**
  + **用途**：美食的名称。
  + **类型**：字符串（String），例如 "烤鸭", "炒鸡肉"。
* **restaurantID**
  + **用途**：美食所在餐厅的ID。
  + **类型**：整数（Integer），例如 1, 1。
* **cuisine**
  + **用途**：美食的菜系。
  + **类型**：字符串（String），例如 "京菜", "川菜"。
* **popularity**
  + **用途**：美食的受欢迎程度。
  + **类型**：整数（Integer），例如 65, 97。

#### 4. **routes**

* **start**
  + **用途**：道路的起点ID。
  + **类型**：整数（Integer），例如 2, 2。
* **end**
  + **用途**：道路的终点ID。
  + **类型**：整数（Integer），例如 0, 1。
* **start\_coordinates**
  + **用途**：起点的坐标。
  + **类型**：整数数组（Array of Integers），例如 [179, 229], [179, 229]。
* **end\_coordinates**
  + **用途**：终点的坐标。
  + **类型**：整数数组（Array of Integers），例如 [237, 137], [140, 107]。
* **midpoint**
  + **用途**：道路拐点的坐标。
  + **类型**：整数数组（Array of Integers），例如 [179, 137], [179, 107]。
* **distance**
  + **用途**：道路的长度。
  + **类型**：整数（Integer），例如 150, 161。
* **walkCongestion**
  + **用途**：道路的步行拥挤度。
  + **类型**：双精度浮点数（Double），例如 0.89, 0.72。
* **bikeCongestion**
  + **用途**：道路的骑行拥挤度。
  + **类型**：双精度浮点数（Double），例如 0.9, -1。
* **electricCongestion**
  + **用途**：道路的电动车拥挤度。
  + **类型**：双精度浮点数（Double），例如 0.96, 0.82。

### 4. 功能分析

#### 4.1 全局变量

**attr 数组**

存储所有的景区信息，包含最多200个景区。

Attraction attr[MAX\_ATTRACTION\_NUM]; // 景区数组

#### 4.2 功能函数

**加载和保存景区信息**

用于将 JSON 文件中的景区数据加载到内存中的 attr 数组，以及将 attr 数组中的数据保存到 JSON 文件中。

void loadAttractionArray(string filename); // 从 JSON 文件加载景区数据  
void saveAttractionArray(string filename); // 将景区数据保存到 JSON 文件

**路线规划相关函数**

这些函数用于计算从起点到终点的最短路径或最短时间，以及处理多点路径规划。

void two\_point\_route\_planning(ALGraph\*, bool); // 两点间的路径规划  
void Multi\_point\_route\_planning(ALGraph\* graph, bool); // 多点间的路径规划  
void DjikstraByTime(ALGraph\* graph, int start, int flag); // 以时间为权重的迪杰斯特拉算法  
void DjikstraByDist(ALGraph\* graph, int start); // 以距离为权重的迪杰斯特拉算法  
void FloydByDist(ALGraph\* graph, int start, int num, int place[]); // 以距离为权重的弗洛伊德算法  
void FloydByTime(ALGraph\* graph, int start, int num, int place[], int flag); // 以时间为权重的弗洛伊德算法

**辅助函数**

这些函数用于路径规划过程中的辅助操作，比如获取路径、保存路径和计算距离等。

int getdis(int num, int place[]); // 计算路径距离  
void savpath(int place[], int num); // 保存路径  
void getpath(int start, int end); // 获取路径  
void printPath(ALGraph\* graph, int start, int end); // 打印路径  
void printMidpoint(ALGraph\* graph, int start, int end); // 打印路径中的所有点和中间点

#### 4.3 数据结构功能和关系

**Attraction 结构体**

Attraction 结构体是景区的顶层数据结构，包含景区的基本信息（如名称、热度、评分等）以及景区内部的详细地图（包括景点、美食和道路信息）。每个 Attraction 实例表示一个具体的景区。

**ALGraph 结构体**

ALGraph 结构体是邻接表结构，用于表示景区内部的地图。它包含一个 VNode 数组，每个 VNode 实例表示一个景点。VNode 实例通过 ArcNode 链表连接，表示景点之间的道路。

**VNode 结构体**

VNode 结构体表示一个景点，包括景点的基本信息（如名称、坐标等）以及指向第一条以该景点为起点的边的指针。通过 VNode 可以访问到以该景点为起点的所有道路信息。

**ArcNode 结构体**

ArcNode 结构体表示景点之间的道路，包括道路的基本信息（如终点ID、长度、拥挤度等）以及指向下一条边的指针。通过 ArcNode 可以构成景点之间的链表结构，表示景点之间的连通性。

**Foods 结构体**

Foods 结构体表示景区内的美食信息，包括美食的基本信息（如名称、餐

厅ID、菜系、受欢迎程度等）。Foods 数组包含在 Attraction 结构体中，用于表示景区内所有美食的信息。

#### 4.4 数据结构之间的具体关系

1. **Attraction** 包含了一个 **ALGraph** 对象，表示该景区的内部地图，和一个 **Foods** 数组，表示该景区内的所有美食。
2. **ALGraph** 包含了一个 **VNode** 数组，表示景区内的所有景点。每个 **VNode** 实例表示一个具体的景点，包括景点的名称、坐标等基本信息。
3. **VNode** 包含了一个指向 **ArcNode** 的指针，表示从该景点出发的所有道路。**ArcNode** 通过链表结构连接，表示道路之间的关系。每个 **ArcNode** 实例表示一条具体的道路，包括道路的终点ID、长度、拥挤度等基本信息。
4. **Foods** 数组包含在 **Attraction** 结构体中，用于表示景区内所有美食的信息。每个 **Foods** 实例表示一种具体的美食，包括美食的名称、餐厅ID、菜系、受欢迎程度等基本信息。

#### 4.5 具体代码实现中的数据结构分析

在代码实现中，这些数据结构通过函数进行操作和处理，下面详细分析各个函数的作用和功能。

**加载景区信息函数**

用于将 JSON 文件中的景区数据加载到内存中的 attr 数组。

void loadAttractionArray(string filename);

**保存景区信息函数**

用于将 attr 数组中的景区数据保存到 JSON 文件中。

void saveAttractionArray(string filename);

### 5. 数据量描述

在本项目中，景区和校园数量为200个，每个景区和校园包含的建筑物（景点、教学楼、办公楼、宿舍楼）数量为20个，其他服务设施数量为50个。景区内美食种类10种，每种美食的数量为20个。景区内道路（边）的数量为200条。实际数据量可能更大，但这些数量设置尽量接近真实场景，便于项目功能实现和优化。

#### 数据量示例

* **景区数量**：200个。
* **景点数量**：20个/景区，总计4000个。
* **服务设施数量**：50个/景区，总计10000个。
* **美食数量**：20个/景区，总计4000个。
* **道路数量**：200条，总计40000条。

## 日记数据结构分析

日记数据结构主要包含以下字段：ID, title, destination, author, rating, popularity, content。这些字段详细描述了一篇日记的各种信息。diaries.json 文件存储了1000篇日记的基础信息，但content字段为空，因为内容通过压缩机制存储在外部文件中，并在需要时通过解压缩机制读取。整个数据结构通过 Diary 结构体来表示，并通过 JSON 文件和外部文件结合使用，实现了高效的数据存储和读取。

### 1. Diary 结构体定义

Diary 结构体定义在diarymanagement.h文件中，用于表示一篇日记。具体定义如下：

struct Diary {  
 int ID;  
 std::string title;  
 std::string destination;  
 std::string author;  
 double rating;  
 int popularity;  
 std::string content;  
};

#### 字段解析

1. **ID**
   * **类型**: int
   * **用途**: 唯一标识一篇日记的ID。
   * **示例**: 1, 2, 3
2. **title**
   * **类型**: std::string
   * **用途**: 日记的标题。
   * **示例**: "探索林 海滩的美", "Lost in the Beauty of Nature"
3. **destination**
   * **类型**: std::string
   * **用途**: 日记中提到的目的地。
   * **示例**: "林 海滩", "David Forest"
4. **author**
   * **类型**: std::string
   * **用途**: 日记的作者。
   * **示例**: "gp", "qnhk"
5. **rating**
   * **类型**: double
   * **用途**: 日记的评分，表示该日记的质量或受欢迎程度。
   * **示例**: 3.2066082191780825, 2.656854945054945
6. **popularity**
   * **类型**: int
   * **用途**: 日记的受欢迎程度，通常基于查看或点赞次数。
   * **示例**: 73, 91
7. **content**
   * **类型**: std::string
   * **用途**: 日记的内容。在diaries.json文件中，该字段为空，因为内容被压缩并存储在外部文件中。
   * **示例**: ""

### 2. diaries.json 文件格式

diaries.json 文件中存储了所有日记的基础信息。文件内容示例如下：

{  
 "diaries": [  
 {  
 "title": "探索林 海滩的美",  
 "ID": 1,  
 "destination": "林 海滩",  
 "author": "gp",  
 "rating": 3.2066082191780825,  
 "popularity": 73,  
 "content": ""  
 },  
 {  
 "title": "Lost in the Beauty of Nature",  
 "ID": 2,  
 "destination": "David Forest",  
 "author": "qnhk",  
 "rating": 2.656854945054945,  
 "popularity": 91,  
 "content": ""  
 }  
 ]  
}

### 3. 压缩和解压缩机制

**内容压缩**:

* 在 DiaryCreate::on\_confirm\_clicked 函数中，日记内容被压缩并存储到二进制文件和编码表JSON文件中，而 content 字段在 diaries.json 文件中为空。
* 具体在本项目中，日记内容压缩的二进制文件存在data文件夹中，data文件夹的具体位置见总体方案设计报告中的项目结构中。

压缩函数compressString

**内容解压缩**:

* 在读取日记内容时，parseDiariesContents 函数会调用 decompressString 函数来解压缩内容，并填充到 Diary 结构体中。

解压缩函数decompressString

### 4. 日记数据结构在代码中的使用

* **存储日记数据**:  
  在 DiaryCreate::on\_confirm\_clicked 函数中，新的日记数据被创建，并通过 compressString 函数压缩内容，然后保存到 diaries.json 文件中。
* **读取日记数据**:  
  在 diaryManagement 类的构造函数中，diaries.json 文件被读取并解析，通过 parseDiariesContents 函数将日记数据填充到 std::vector<Diary> 中。

### 5. diaryManagement 类的解析函数

std::vector<Diary> diaryManagement::parseDiariesContents(cJSON \*diaries)
  
{
  
 std::vector<Diary> result;
  
  
 cJSON\* diaryArray = cJSON\_GetObjectItem(diaries, "diaries");
  
 if (!diaryArray) {
  
 qDebug() << "Invalid JSON format. 'diaries' array not found.";
  
 return result;
  
 }
  
  
 cJSON\* entry = diaryArray->child;
  
 while (entry != NULL) {
  
 Diary diary = {
  
 cJSON\_GetObjectItem(entry, "ID")->valueint,
  
 cJSON\_GetObjectItem(entry, "title")->valuestring,
  
 cJSON\_GetObjectItem(entry, "destination")->valuestring,
  
 cJSON\_GetObjectItem(entry, "author")->valuestring,
  
 cJSON\_GetObjectItem(entry, "rating")->valuedouble,
  
 cJSON\_GetObjectItem(entry, "popularity")->valueint,
  
 decompressString(cJSON\_GetObjectItem(entry, "ID")->valueint)
  
 };
  
 result.push\_back(diary);
  
 entry = entry->next;
  
 }
  
  
 return result;
  
}

### 6. 数据量描述

在本项目中，diaries.json 文件存储了1000篇日记的基础信息（仍支持继续添加）。每篇日记包含如下字段：

* ID: 唯一标识
* title: 标题
* destination: 目的地
* author: 作者
* rating: 评分
* popularity: 受欢迎程度
* content: 内容（在json文件中为空，通过外部文件存储压缩内容）

#### 数据量示例

假设每篇日记的内容平均为1000个字符，采用哈夫曼压缩算法，压缩率约为40%，则每篇日记的压缩后大小约为400个字符。存储1000篇日记的内容约为400,000字符（400KB）。而diaries.json文件主要存储日记的元数据信息，大小约为数十KB。

## 路线规划与推荐功能的数据结构分析

### 1. 路线规划相关结构体分析

#### 1.1 Coordinate 结构体

struct Coordinate {
  
 int x;
  
 int y;
  
 Coordinate\* next;
  
};

**功能**：表示路线中的坐标点。

**字段**：

* x：横坐标，类型为 int。
* y：纵坐标，类型为 int。
* next：指向下一个坐标点，类型为 Coordinate\*。

#### 1.2 全局变量

extern int dist[MAX\_VERTEX\_NUM];
  
extern int prevNode[MAX\_VERTEX\_NUM];
  
extern int anspath[MAX\_VERTEX\_NUM];
  
extern int idx;

**功能**：存储路径规划中的中间结果和最终结果。

**字段**：

* dist：从起点到各个顶点的最短距离，类型为 int[MAX\_VERTEX\_NUM]。
* prevNode：路径中前一个节点的索引，类型为 int[MAX\_VERTEX\_NUM]。
* anspath：存储最终路径的节点索引，类型为 int[MAX\_VERTEX\_NUM]。
* idx：路径中节点的数量，类型为 int。

### 2. 推荐功能相关结构体函数分析

#### 2.1 destinationRecommend 结构体

private:
  
 QStringListModel \*m\_model;
  
 int attrIndex[MAX\_ATTRACTION\_NUM];

**功能**：表示景区推荐界面及其功能。

**字段**：

* m\_model：数据模型，用于显示推荐的景区列表，类型为 QStringListModel\*。
* attrIndex：存储推荐景区的索引数组，类型为 int[MAX\_ATTRACTION\_NUM]。

### 3. 数据结构关系

#### 3.1 Coordinate 与 ArcNode

**关系**：Coordinate 结构体表示路径中的坐标点，ArcNode 结构体表示道路，通过路径规划函数计算出路径中的所有坐标点。

#### 3.2 ALGraph 与 VNode 和 ArcNode

**关系**：ALGraph 结构体表示景区的内部地图，包含所有景点（VNode）和道路（ArcNode），通过邻接表存储。

#### 3.3 destinationRecommend 与 Attraction

**关系**：destinationRecommend 结构体用于推荐景区，基于 Attraction 结构体存储的景区信息进行推荐和搜索。

### 4. 数据量描述

* 和前面一致