集合运算系统

1. 题目说明与分析
2. 题目说明

以单链表的形式表示一个集合，试编写算法实现：

（1）求出两个集合A和B的交集，即 A∩B；

（2）求出两个集合A和B的差集，即 A- B；

（3）求出两个集合A和B的并集，即 A∪B。

要求：A和B集合是由任意输入的数据构建的单链表，不是程序定死的。

1. 题目分析

根据上述题目描述可知：

（1）这是一个实现集合运算的程序，需要使用单链表数据结构来存储集合元素。因为输入的集合元素可以是任意类型数据，例如数字、中文、英文或多种混合的类型。所以存储集合元素变量类型应定义为字符串类型。

**(2)** 程序需要实现集合的交集、差集、并集等基本运算。程序中应包含数据录入、数据显示、集合运算等功能，可以划分为独立的模块，并使用自定义函数来实现。

**(3)** 为了方便用户操作，需要设计一个用户交互界面，允许用户输入集合元素，并选择需要进行的集合运算。

(4) 需要在部分输入场景考虑对输入数据进行验证，以防止用户输入错误的数据导致程序运行错误。

(5) 因为要求输入数据不能定死，所以本题不考虑使用文本文件来保存集合元素，使用即用即存即算的设计思路。

(6) 可以考虑在用户界面中增加一些辅助功能，例如显示集合元素数量、批量增加集合元素、修改和删除集合元素、清空集合等。

(7) 可以进一步扩展功能，例如实现其他集合运算，或者实现更复杂的数据结构，以提高运算效率。

* **单链表数据结构:** 单链表是一种线性数据结构，由一系列节点构成，每个节点包含数据域和指向下一个节点的指针域。
* **集合运算:** 集合运算主要包括交集、差集、并集等操作。
* **用户交互界面设计:** 需要设计一个简单的用户界面，方便用户输入集合元素并进行集合运算。

二、系统设计

1. 理论基础

**单链表**：单链表是一种线性数据结构，每个节点包含数据域和指向下一个节点的指针域。

**集合运算**：集合运算包括以下几种操作：

* **交集:** 交集包含在两个集合中所有元素。
* **差集:** 差集包含在一个集合中但不包含在另一个集合中的所有元素。
* **并集:** 并集包含在两个集合中所有元素，包括重复的元素。

2. 数据类型定义

使用结构体 Node 来表示单链表中的节点，包含数据域 data 和指向下一个节点的指针域 next。

解释

struct Node {

string data;

Node\* next=NULL;

};

使用类 LinkedList 来表示单链表，包含头节点指针 head 和链表长度 Length。

解释

class LinkedList {

private:

Node\* head = NULL;

int Length = 0;

public:

*// ...*

};

3. 函数定义及说明

* insert(const string& value)：在链表头部插入一个新节点。
* deletedata(Node\* finded)：删除链表中指定节点。
* display() const：显示链表中所有元素。
* getHead() const：返回链表的头节点指针。
* length() const：返回链表长度。
* finddata(string value) const：查找链表中是否包含特定数据，若包含返回节点指针，否则返回 NULL。
* intersection(LinkedList& A, LinkedList& B)：计算两个链表的交集。
* difference(LinkedList& A, LinkedList& B)：计算两个链表的差集。
* unionSet(LinkedList& A, LinkedList& B)：计算两个链表的并集。

4. 系统模块结构

系统模块结构如下：

1. **数据结构模块:** 包含 Node 和 LinkedList 类定义。
2. **集合运算模块:** 包含实现集合运算的函数。
3. **用户交互界面模块:** 包含用户界面显示和交互功能。

三、系统实现

核心功能函数的程序流程图

1. intersection 函数流程图

解释

graph LR

A[链表 A] --> B[链表 B]

B --> C[遍历 B]

C --> D{数据相同?}

D -- 否 --> C

D -- 是 --> E[插入交集]

E --> A

A --> F[遍历 A]

F --> C

2. difference 函数流程图

解释

graph LR

A[链表 A] --> B[链表 B]

B --> C[遍历 B]

C --> D{数据相同?}

D -- 否 --> C

D -- 是 --> F[继续遍历 B]

F --> C

C --> E[遍历 A]

E --> G{数据未找到?}

G -- 否 --> E

G -- 是 --> H[插入差集]

H --> A

A --> I[遍历 A]

I --> E

3. unionSet 函数流程图

解释

graph LR

A[链表 A] --> B[遍历 A]

B --> C[插入 A]

C --> A

A --> D[链表 B]

D --> E[遍历 B]

E --> F[遍历 A]

F --> G{数据相同?}

G -- 否 --> H[插入 B]

H --> D

G -- 是 --> E

D --> I[遍历 B]

I --> E

四、程序测试与结果分析

程序运行步骤

1. 运行程序。
2. 输入集合元素，可以使用单个添加或批量添加的方式。
3. 选择需要执行的集合运算。
4. 程序会显示计算结果。

测试用例输入格式及说明

* 输入集合元素：以字符串形式输入，例如 “apple”, “banana”, “orange”。
* 输入集合运算：选择菜单选项对应不同的运算。

运行结果及说明

* 交集运算：输入两个集合 A 和 B，分别包含 “apple”, “banana”, “orange” 和 “banana”, “grape”, “apple”。交集结果为 “apple”, “banana”。
* 差集运算：输入两个集合 A 和 B，分别包含 “apple”, “banana”, “orange” 和 “banana”, “grape”, “apple”。 A - B 差集结果为 “orange”，B - A 差集结果为 “grape”。
* 并集运算：输入两个集合 A 和 B，分别包含 “apple”, “banana”, “orange” 和 “banana”, “grape”, “apple”。并集结果为 “apple”, “banana”, “orange”, “grape”。

程序运行截图

五、总结

系统优势

* 代码结构清晰，易于理解和维护。
* 功能完善，能够实现基本的集合运算。
* 用户界面简单易用，方便用户进行操作。

系统不足

* 缺乏对输入数据类型和格式的验证，可能会导致程序运行错误。
* 缺乏对内存泄漏的处理机制，可能会导致程序运行效率低下。
* 用户界面设计较为简陋，可以进一步完善。

设想和可扩展功能

* 增加对数据类型和格式的验证，提高程序的健壮性。
* 增加对内存泄漏的处理机制，提高程序的运行效率。
* 设计更加完善的用户界面，增加更多功能，例如撤销、重做等操作。
* 扩展集合运算功能，例如对称差集、补集等