项目说明文档

数据结构课程设计

——两个有序链表序列的交集

作 者 姓 名： 张翔

学 号： 2352985

指 导 教 师： 张颖

学院、 专业： 计算机科学与技术学院 软件工程

同济大学

Tongji University

二〇二四 年 十二 月 七 日

# 1项目分析

## 1.1项目背景分析

在计算机科学和实际应用中，链表是一种基础的数据结构，广泛用于存储和操作动态数据。在数据处理中，找到两个有序集合（链表）的交集是一个常见问题。例如：

(1)数据库查询中，需要找出满足多个条件的数据（类似集合交集操作）。

(2)集合运算中，交集操作可用于分析重叠数据。

在本项目中，两个输入链表和都是按非降序排列的（即从小到大排序），可以利用这一特性设计高效算法，避免冗余计算。

## 1.2项目需求分析

本项目需要接受两个非降序排列的链表S1和S2作为输入，并构造一个包含它们交集的新链表 S3​，保持非降序排列。链表节点的数据类型为整数，且输入链表中的元素互不重复。当S1和 S2无交集时，输出为空链表。程序应具有高效性，利用输入链表的有序性减少不必要的计算，处理任意链表为空的边界情况，并保证输出结果的准确性和清晰性。

## 1.3项目功能分析

程序分为三个主要功能模块：输入模块用于构造链表S1和S2​，支持用户手动输入或从文件加载数据；交集计算模块通过逐一比较 S1和 S2的节点值，生成结果链表 S3，确保操作高效且保持非降序排列；输出模块用于显示交集链表的所有元素，若交集为空则提供提示信息。

# 2项目设计

## 2.1数据结构设计

基于项目分析，本项目选择链表作为数据结构而不是数组，主要原因如下:

(1)动态大小需求：链表可以动态地分配内存;

(2)内存分配灵活性：链表的内存分配比较灵活，可以根据需求动态分配，而数组需要一次性分配固定大小的内存空间；

(3)避免数组扩展的开销：使用数组可能需要额外的扩展和拷贝操作，而链表避免了这种开销；

## 2.2 MyList类的设计

### 2.2.1概述

MyList是一个基于模板的链表类，用于实现通用的链表操作。它通过MyLinkNode<Type>\* head 维护链表的头节点，支持多种功能，包括动态创建链表（可通过指定终止标志或直接传入头节点进行构造）、遍历打印链表内容，以及获取链表头节点指针等。默认构造函数初始化为空链表，析构函数负责释放内存以防止泄漏。通过模板参数Type，MyList可存储任意类型的数据，适用于动态数据存储及操作的场景，具有良好的扩展性和适配性。

### 2.2.2类定义

template <typename Type>

class MyList

{

private:

MyLinkNode<Type>\* head;

public:

MyList() : head(nullptr) {}

~MyList();

void createList(Type& end);

void createList(MyLinkNode<Type>\* \_head);

void printList();

MyLinkNode<Type>\* getHead();

};

## 2.3项目主体架构设计

|  |
| --- |
|  |

2.3.1项目主体架构流程图

# 3项目功能实现

## 3.1输入链表功能的实现

### 3.1.1输入链表功能实现思路

这段代码实现了链表的动态创建功能，允许用户输入一组数据以构建链表，并通过指定的结束标志（end）结束输入，具体实现思路如下:

函数首先定义一个临时变量num用于接收用户输入，并初始化指针current指向链表头节点 head。在循环中，每次从标准输入读取数据，并检查是否等于结束标志end，如果是则终止输入并清空输入流。对于每个有效数据，动态分配内存创建一个新节点，并根据链表是否为空执行不同的操作：若链表为空，新节点成为链表头节点；否则，将当前节点的link指针指向新节点，并移动 current指针到新节点。若内存分配失败，则输出错误信息并中止操作。该方法确保了链表的动态扩展，同时处理了内存分配失败的情况以提高代码的可靠性。

### 3.1.2输入链表功能核心代码

template <typename Type>

void MyList<Type>::createList(Type& end)

{

Type num = end;

MyLinkNode<Type>\* current = head;

while (std::cin >> num) {

if (num == end) {

std::cin.ignore(INT\_MAX, '\n');

break;

}

MyLinkNode<Type>\* newNode = new(std::nothrow) MyLinkNode<Type>(num);

if (newNode != nullptr) {

if (head == nullptr) {

head = newNode;

current = head;

}

else {

current->link = newNode;

current = newNode;

}

}

else {

std::cerr << "Memory allocation failed for MyLinkNode." << std::endl;

break;

}

}

}

## 3.2查找链表交集功能的实现

### 3.2.1查找链表交集功能实现思路

这段代码实现了两个有序链表的交集计算功能，其核心思想是利用链表的有序性，通过双指针逐步遍历两个链表并比较节点值，具体思路如下:

初始化两个指针p1和p2分别指向链表l1和l2的头节点，并用dummy和result维护交集链表的头节点和尾节点。在遍历过程中，如果p1和p2的数据相等，则创建一个新节点加入交集链表，并同时移动p1和p2指针；如果p1的数据小于p2，则移动p1；否则移动p2，从而保证比较的有效性和效率。遍历结束后返回交集链表的头节点。代码通过检查内存分配结果来确保程序的健壮性，处理了节点数据重复及链表长度不同的情况。

### 3.2.2查找链表交集功能核心代码

template <typename Type>

MyLinkNode<Type>\* findIntersection(MyLinkNode<Type>\* l1, MyLinkNode<Type>\* l2)

{

MyLinkNode<Type>\* p1 = l1;

MyLinkNode<Type>\* p2 = l2;

MyLinkNode<Type>\* dummy = nullptr;

MyLinkNode<Type>\* result = nullptr;

while (p1 && p2) {

if (p1->data == p2->data) {

MyLinkNode<Type>\* newNode = new(std::nothrow) MyLinkNode<Type>(p1->data);

if (newNode != nullptr) {

if (result == nullptr) {

result = newNode;

dummy = result;

}

else {

dummy->link = newNode;

dummy = newNode;

}

p1 = p1->link;

p2 = p2->link;

}

else {

std::cerr << "Memory allocation failed for MyLinkNode." << std::endl;

}

}

else if (p1->data < p2->data) {

p1 = p1->link;

}

else if (p1->data > p2->data) {

p2 = p2->link;

}

}

return result;

}

## 3.3异常处理功能的实现

### 3.3.1动态内存申请失败的异常处理

在进行动态内存申请时，程序使用new(std::nothrow)来尝试分配内存。new(std::nothrow)在分配内存失败时不会引发异常，而是返回一个空指针（nullptr），代码检查指针是否为空指针，如果为空指针，意味着内存分配失败，这时程序将执行以下操作：

(1)向标准错误流std::cerr输出一条错误消息"Error: Memory allocation failed."；

(2)调用exit函数，返回错误码-1，用于指示内存分配错误，并导致程序退出。

### 3.3.2输入非法的异常处理

程序通过调用inputInteger函数输入电网节点个数和电网节点之间的距离。inputInteger函数用于获取用户输入的整数，同时限制输入必须在指定的范围内，函数的代码如下：

int inputInteger(int lowerLimit, int upperLimit, const char\* prompt)

{

std::cout << ">>> " << "请输入" << prompt << " 整数范围: [" << lowerLimit << "~" << upperLimit << "]: ";

int input;

while (true) {

std::cin >> input;

if (std::cin.good() && input >= lowerLimit && input <= upperLimit) {

std::cin.clear();

std::cin.ignore(INT\_MAX, '\n');

std::cout << std::endl;

return input;

}

else {

std::cerr << ">>> " << prompt << "输入不合法，请重新输入！" << std::endl;

std::cin.clear();

std::cin.ignore(INT\_MAX, '\n');

}

}

}

# 4项目测试

|  |
| --- |
|  |

4.1一般情况测试示例

|  |
| --- |
|  |

4.2交集为空的情况测试示例

|  |
| --- |
|  |

4.3完全相交的情况测试示例

|  |
| --- |
|  |

4.4其中一个序列完全属于交集的情况测试示例

# 5集成开发环境与编译运行环境

Windows系统：Windows 11 x64

Windows集成开发环境：Microsoft Visual Studio 2022 (Release模式)

Windows编译运行环境：本项目适用于x86架构和x64架构