|  |  |
| --- | --- |
| 姓名 | 单禹嘉 |
| 学号 | 2023215177 |

|  |  |
| --- | --- |
| 实验成绩 |  |



华中师范大学计算机学院

实验报告书

课程名称：数据结构

主讲教师：沈显君

课程编号：

班级：2306

1. 实验目的

目的：

通过利用线性链表数据结构，模拟并实现两个整数集合的交集、并集和差集运算。通过实验，掌握线性链表在集合运算中的基本应用。

1. 实验基本内容

1. 需求分析： 设计一个系统，能够基于链表实现两个集合的交集、并集和差集运算。用户输入两个集合，系统输出相应运算结果。

2. 抽象数据类型定义： 定义用于表示集合的链表结构，并实现集合的基本运算，如并集、交集和差集。

3. 系统流程设计： 系统接收用户输入的集合数据，计算交集、并集、差集，输出运算结果。

4. 链表操作：

• 初始化链表（表示集合）。

• 实现并集、交集和差集的运算。

• 输出运算结果。

1. 实验环境

VSCode

1. 系统描述

本系统利用双向循环链表存储学生信息，支持按学生总成绩从高到低排序（若总成绩相同，则按所用时间升序排列）。系统旨在高效管理学生数据，完成排序后输出结果。系统包括以下模块：

1. 数据存储模块：利用双向循环链表存储学生信息。

2. 排序模块：采用快速排序算法实现链表排序。

3. 数据展示模块：打印学生信息供用户查看。

1. 需求分析

系统需满足以下需求：

1. 基本功能：

• 从文件中读取学生信息，包括学号、姓名、三门课程成绩及所用时间。

• 计算每位学生的总成绩。

• 使用双向循环链表存储学生信息。

• 按照总成绩降序排序，若总成绩相同，按所用时间升序排列。

2. 扩展功能：

• 支持动态插入和删除学生信息。

• 打印学生信息的排序结果。

3. 性能需求：

• 快速排序的时间复杂度为 O(n \log n)。

• 占用内存应尽量少，适合多节点操作

1. 详细概要设计

系统基于以下模块设计：

1. 数据结构设计

• 每个学生信息用 Student 结构体表示，包含学号、姓名、三门课程成绩、总成绩、所用时间，以及双向指针（prev 和 next）。

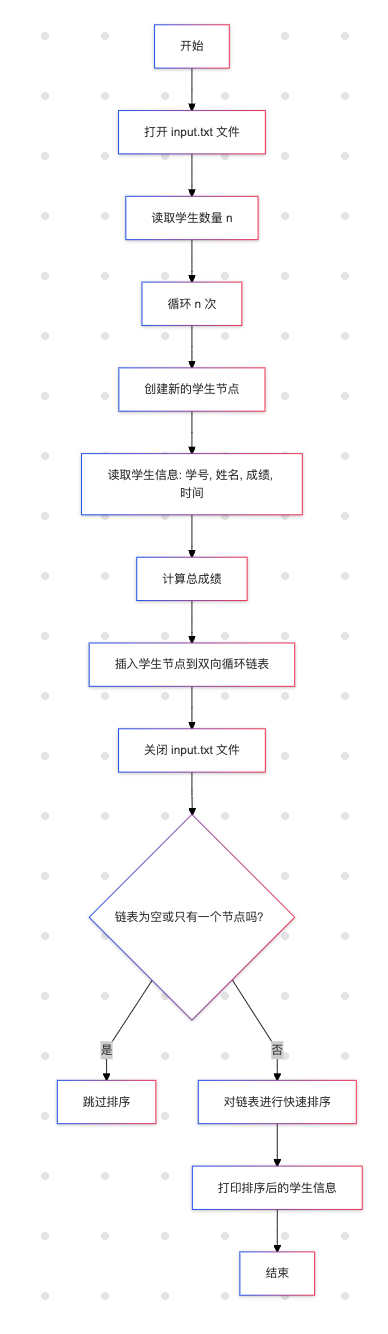
2. 功能设计

• 链表构建：通过双向循环链表插入学生节点。

• 排序：实现快速排序对链表进行排序。

• 输出结果：遍历链表输出学生信息。

3.系统功能模块图



1. 系统详细设计原理与过程

1. 双向循环链表实现

链表通过以下步骤实现：

• 初始化空链表。

• 插入节点时，调整前驱和后继指针，确保链表首尾相连。

• 打印链表时从头节点开始，循环访问每个节点。

2. 快速排序实现

• 设置最后一个节点为基准点（pivot），进行分区。

• 递归调用排序链表的前半部分和后半部分，直到完成排序。

• 在分区过程中，利用指针交换节点数据。

1. 系统运行结果与分析

输入文件 input.txt 示例：

5

20230001 ZhangSan 95 89 90 120

20230002 LiSi 78 82 88 110

20230003 WangWu 95 89 90 115

20230004 ZhaoLiu 65 70 72 100

20230005 QianQi 95 89 90 110

输出结果：

排序后的学生信息：

学号: 20230001, 姓名: ZhangSan, 高等数学: 95, 数据结构: 89, C++: 90, 总分: 274, 时间: 120

学号: 20230003, 姓名: WangWu, 高等数学: 95, 数据结构: 89, C++: 90, 总分: 274, 时间: 115

学号: 20230005, 姓名: QianQi, 高等数学: 95, 数据结构: 89, C++: 90, 总分: 274, 时间: 110

学号: 20230002, 姓名: LiSi, 高等数学: 78, 数据结构: 82, C++: 88, 总分: 248, 时间: 110

学号: 20230004, 姓名: ZhaoLiu, 高等数学: 65, 数据结构: 70, C++: 72, 总分: 207, 时间: 100

1. 系统使用说明

1. 准备好 input.txt 文件，按照规定格式存储学生数据。

2. 运行程序后，结果直接显示在控制台。

3. 如果需要更改数据，直接编辑 input.txt 文件并重新运行程序。

1. 系统调试经验与心得

1. 调试经验：

• 检查链表的正确性：初期调试过程中，链表循环指针设置错误，导致死循环。通过调试，确保 next 和 prev 指针正确指向。

• 数据交换时，使用 strcpy 和 memcpy 避免指针操作导致的错误。

2. 心得体会：

• 双向循环链表适合动态数据操作场景，与数组相比更灵活。

• 快速排序与链表结合较复杂，但优化排序逻辑后运行效率显著提升。

• 通过本实验，深入理解了链表与排序算法的应用。

1. 附录（源代码）与说明

#include <iostream>

#include <fstream>

#include <string>

#include <cstring>

#include <vector>

using namespace std;

//Mermain[]

struct Student {

char id[9]; // 学号（8位字符）

char name[11]; // 姓名（10位字符）

int scores[3]; // 各科成绩（例如：高等数学、数据结构、C++）

int totalScore; // 总成绩

int time; // 所用时间（整型）

Student\* prev; // 前驱指针

Student\* next; // 后继指针

};

class DoublyCircularLinkedList {

public:

Student\* head; // 链表头指针

DoublyCircularLinkedList() : head(nullptr) {}

// 插入学生信息节点

void insert(Student\* newStudent) {

if (!head) {

head = newStudent;

newStudent->next = head;

newStudent->prev = head;

} else {

Student\* tail = head->prev; // 获取尾部节点

tail->next = newStudent;

newStudent->prev = tail;

newStudent->next = head;

head->prev = newStudent;

}

}

// 打印链表中的学生信息

void printList() {

if (!head) return;

Student\* temp = head;

do {

cout << "学号: " << temp->id << ", 姓名: " << temp->name

<< ", 高等数学: " << temp->scores[0]

<< ", 数据结构: " << temp->scores[1]

<< ", C++: " << temp->scores[2]

<< ", 总分: " << temp->totalScore

<< ", 时间: " << temp->time << endl;

temp = temp->next;

} while (temp!= head);

}

};

// 计算总成绩

void calculateTotalScore(Student\* student) {

student->totalScore = student->scores[0] + student->scores[1] + student->scores[2];

}

// 交换两个节点的数据

void swap(Student\* a, Student\* b) {

char tempId[9], tempName[11];

int tempScores[3], tempTotalScore, tempTime;

strcpy(tempId, a->id);

strcpy(tempName, a->name);

memcpy(tempScores, a->scores, sizeof(a->scores));

tempTotalScore = a->totalScore;

tempTime = a->time;

strcpy(a->id, b->id);

strcpy(a->name, b->name);

memcpy(a->scores, b->scores, sizeof(b->scores));

a->totalScore = b->totalScore;

a->time = b->time;

strcpy(b->id, tempId);

strcpy(b->name, tempName);

memcpy(b->scores, tempScores, sizeof(tempScores));

b->totalScore = tempTotalScore;

b->time = tempTime;

}

// 快速排序的分区函数

Student\* partition(Student\* low, Student\* high) {

int pivotScore = high->totalScore;

int pivotTime = high->time;

Student\* i = low->prev;

for (Student\* j = low; j!= high; j = j->next) {

// 按总分降序，如果总分相同，按时间升序

if (j->totalScore > pivotScore || (j->totalScore == pivotScore && j->time < pivotTime)) {

i = (i == nullptr)? low : i->next;

swap(i, j);

}

}

i = (i == nullptr)? low : i->next;

swap(i, high);

return i;

}

// 快速排序函数

void quickSort(Student\* low, Student\* high) {

if (high!= nullptr && low!= high && low!= high->next) {

Student\* pivot = partition(low, high);

quickSort(low, pivot->prev);

quickSort(pivot->next, high);

}

}

int main() {

int n;

DoublyCircularLinkedList list;

ifstream inFile("input.txt"); // 假设数据文件名为 input.txt

if (!inFile) {

cerr << "无法打开文件" << endl;

return 1;

}

inFile >> n;

// 输入学生信息

for (int i = 0; i < n; ++i) {

Student\* newStudent = new Student;

inFile >> newStudent->id >> newStudent->name

>> newStudent->scores[0] >> newStudent->scores[1] >> newStudent->scores[2]

>> newStudent->time;

newStudent->prev = newStudent->next = nullptr;

// 计算总分并插入链表

calculateTotalScore(newStudent);

list.insert(newStudent);

}

inFile.close();

// 排序

if (list.head && list.head->next!= list.head) { // 确保链表不为空且有多个节点

quickSort(list.head, list.head->prev);

}

// 输出排序后的学生信息

cout << "\n排序后的学生信息：" << endl;

list.printList();

return 0;

}