实验报告电子版

**数据结构实训**

**——多维数字信号的线性表表示及操作**

姓 名： 江一诺

学 号： 2220220821

指导老师： 韩凤

实验日期： 2024年4月3日

**大连海事大学电子信息工程专业**

**Academic Honesty Violations in Practice of Data Structures (PDS)**

In **PDS**, the MINIMUM penalty recommended for a violation of the Academic Honesty Policy will be a ZERO ON THE ASSIGNMENT, PROJECT or EXAM and a LOWERING OF YOUR FINAL GRADE for below what is otherwise earned. You may NOT withdraw from the course if found guilty. Some examples of academic misconduct in **PDS** include but are not limited to the following actions:

1. Picking up and using or discarding another student's written or computer output;

2. Using the computer account of another student;

3. Representing as one's own the work of another on assignments, quizzes, and projects;

4. Giving another student a copy of one's work on an assignment before the due date.

5. Copying work from online resources (Baidu,Chegg, google forums, etc.)

6. Posting work to online resources where other students can view your work.

All submissions will be checked for similarity. This code will check each submission for similarity between other student submissions, past student submissions, the solution manual, and online resources and postings. If your submission is flagged for a high level of similarity, it will be turned in for an academic honesty violation if deemed appropriate.

NOTE: Changing variable names, adding comments, or spacing will l result in a violation.

1. **实验目的**

1、深入理解线性表的存储结构及操作的基本原理和方法；2、掌握线性表的各种操作（建立、插入、删除等）的实现方法；3、能够实现多维信号的简单运算和操作，如均值滤波、上采样和下采样等。

1. **实验内容与要求**

（1）为一个不定长多维数字信号设计一个数据结构（形如下图示） ，并实现对其进行基本操作，包括初始化、求数据元素个数操作、插入、删除、取数据元素、判非空操作。

（2）能够实现多维信号的简单运算和操作（数字信号处理课讲授的基本方法），如均值滤波、上采样和下采样等*。*

## 3. 算法描述与流程

### 算法描述：

### （1）基本数据结构

**节点结构(Node)**：

三个浮点型坐标值：x, y, z

双向指针：prev, next

**信号链表(SignalList)**：

虚拟头节点：dummyHead

虚拟尾节点：dummyTail

大小计数器：size

### （2）初始化 (init)

init(list)  
输入：信号链表指针list  
输出：无  
步骤：  
1. 为虚拟头节点和尾节点分配内存  
2. 设置头节点的prev为NULL，next指向尾节点  
3. 设置尾节点的prev指向头节点，next为NULL  
4. 初始化链表大小为0

### （3）插入节点 (insert)

insert(list, index, x, y, z)  
输入：链表指针list，插入位置index，坐标值x,y,z  
输出：无  
步骤：  
1. 检查索引是否合法，不合法则直接返回  
2. 从虚拟头节点开始，找到第index个位置前面的节点prev  
3. 创建新节点，赋值x,y,z坐标  
4. 调整指针，完成双向链接：  
 - 新节点的next指向prev的next  
 - 新节点的prev指向prev  
 - prev的next的prev指向新节点  
 - prev的next指向新节点  
5. 链表大小加1

### （4）删除节点 (removeAt)

removeAt(list, index)  
输入：链表指针list，删除位置index  
输出：无  
步骤：  
1. 检查索引是否合法，不合法则直接返回  
2. 找到第index个位置前面的节点prev  
3. 获取要删除的节点toDelete  
4. 调整指针，跳过要删除的节点：  
 - prev的next指向toDelete的next  
 - toDelete的next的prev指向prev  
5. 释放被删除节点内存  
6. 链表大小减1

### （5）均值滤波算法(smooth)

算法：smooth(list)  
输入：信号链表指针list  
输出：无  
步骤：  
1. 如果链表大小小于3，直接返回  
2. 从第二个节点开始遍历到倒数第二个节点  
3. 对每个节点，获取其前驱和后继节点  
4. 对三维坐标分别进行均值计算：  
 - 令当前节点的x = (前驱x + 当前x + 后继x)/3  
 - 令当前节点的y = (前驱y + 当前y + 后继y)/3  
 - 令当前节点的z = (前驱z + 当前z + 后继z)/3  
5. 移动到下一个节点

### （6）上采样算法(upsample)

算法：upsample(list)  
输入：信号链表指针list  
输出：无  
步骤：  
1. 从第一个实际节点开始遍历  
2. 对每个节点，只要当前节点和下一个节点都不是虚拟尾节点：  
 - 获取下一个节点next  
 - 在当前节点和下一个节点之间插入一个新节点，其坐标为两节点坐标的平均值  
 - 移动到next节点继续处理

### （7）下采样算法(downsample)

算法：downsample(list)  
输入：信号链表指针list  
输出：无  
步骤：  
1. 初始化计数器count为0  
2. 从第一个实际节点开始遍历  
3. 对每个节点，记录下一个节点为next  
4. 如果count为奇数：  
 - 删除当前节点（调整前后节点的链接关系）  
 - 释放当前节点内存  
 - 链表大小减1  
5. 计数器count加1  
6. 移动到next节点继续处理

### （8）从文件加载数据 (loadFromFile)

loadFromFile(list, filename)  
输入：链表指针list，文件名filename  
输出：布尔值表示成功与否  
步骤：  
1. 打开文件，如果失败返回false  
2. 清空当前链表  
3. 循环读取文件每一行：  
 - 如果匹配到节点索引，继续读取该节点的x,y,z坐标  
 - 将读取到的坐标插入到链表末尾  
4. 关闭文件并返回true

### （9）保存数据到文件 (saveToFile)

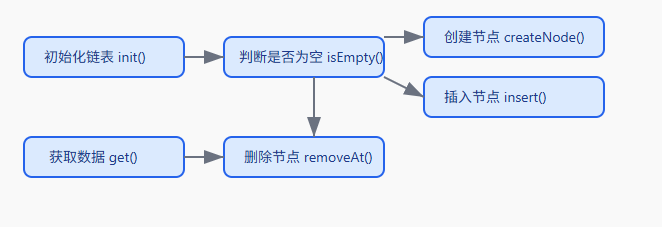
saveToFile(list, filename)  
输入：链表指针list，文件名filename  
输出：布尔值表示成功与否  
步骤：  
1. 创建文件，如果失败返回false  
2. 初始化索引index为1  
3. 从第一个实际节点开始遍历  
4. 对每个节点，将索引和坐标写入文件  
5. 索引递增  
6. 关闭文件并返回true

### （10）主函数

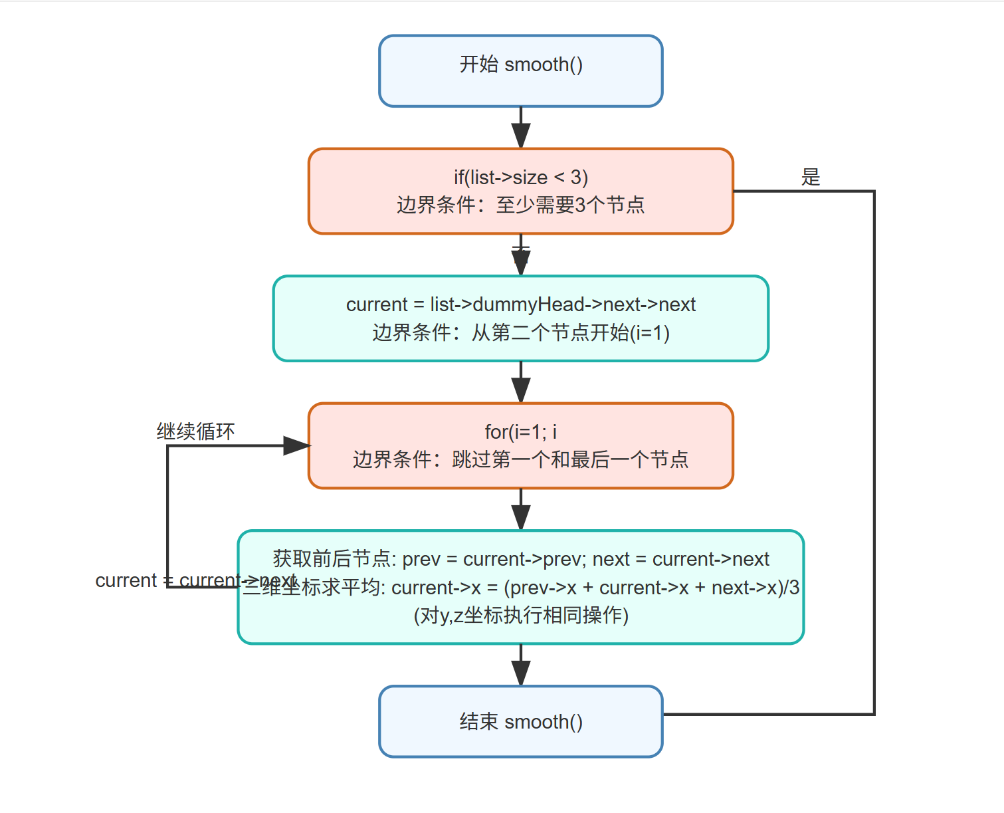
1. 创建并初始化信号链表  
2. 从data.txt文件加载原始信号数据  
3. 若加载成功，执行以下处理：  
 a. 打印原始信号  
 b. 对信号进行均值滤波处理  
 c. 打印滤波后的信号并保存到filtered\_signal.txt  
 d. 对信号进行上采样处理  
 e. 打印上采样后的信号并保存到upsampled\_signal.txt  
 f. 对信号进行下采样处理  
 g. 打印下采样后的信号并保存到downsampled\_signal.txt  
4. 在信号的位置3插入一个新节点(1.0, 2.0, 3.0)  
5. 打印插入新节点后的信号并检查是否为空  
6. 清空信号列表  
7. 打印清空后的信号并检查是否为空  
8. 释放虚拟头尾节点内存  
9. 程序结束

### 流程图

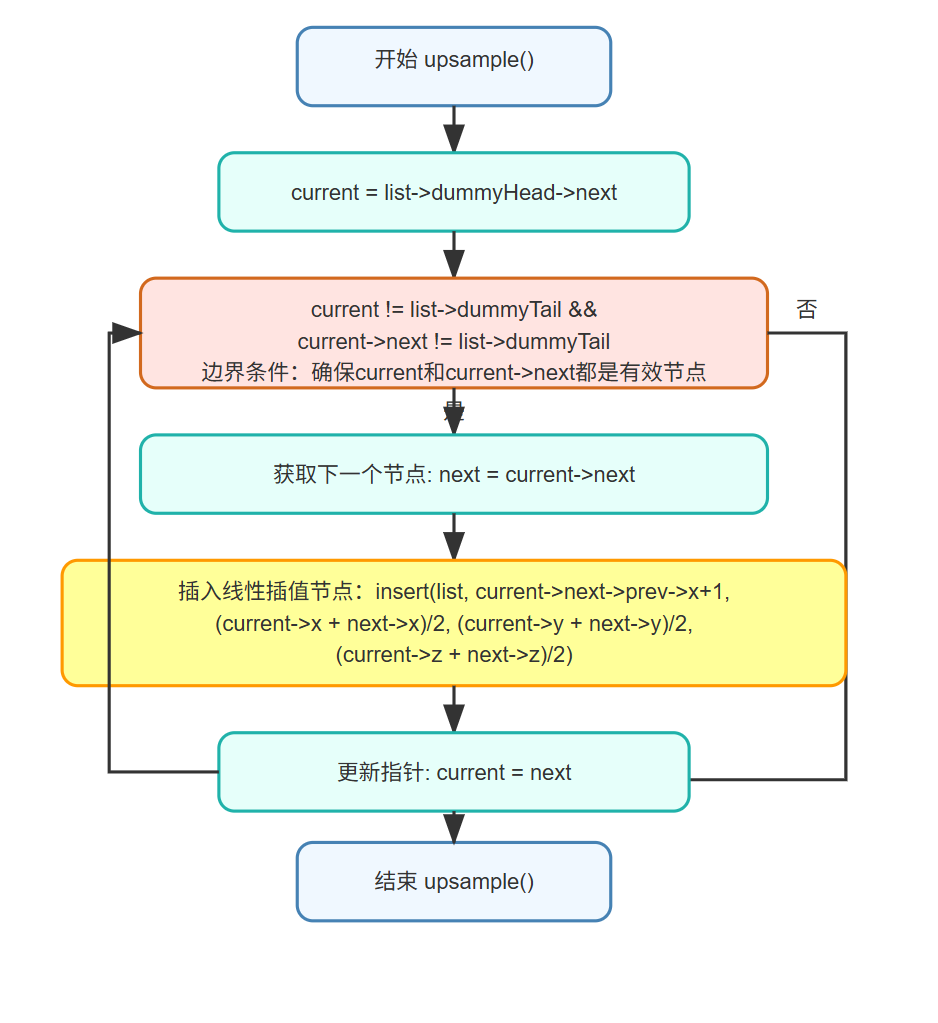
基本数据结构：



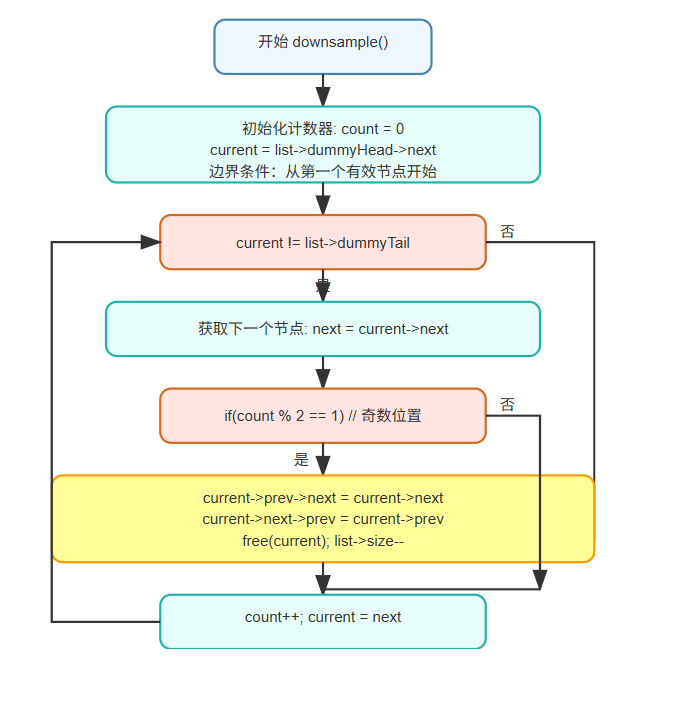
均值滤波处理：



上采样处理：



下采样处理：



1. **算法实现**

#include <stdio.h>

#include <stdlib.h>

#include <stdbool.h>

*// 定义数据类型*

typedef float ElemType;

*// 三维信号节点结构*

typedef struct Node {

    ElemType x, y, z;   *// 三维坐标*

    struct Node \*prev;  *// 前驱指针*

    struct Node \*next;  *// 后继指针*

} Node;

*// 带虚拟头尾节点的双向链表*

typedef struct {

    Node \*dummyHead;

    Node \*dummyTail;

    int size;

} SignalList;

*/\* 初始化链表 \*/*

void init(SignalList \*list) {

    list->dummyHead = (Node\*)malloc(sizeof(Node));*//动态分配内存*

    list->dummyTail = (Node\*)malloc(sizeof(Node));

    list->dummyHead->prev = NULL;

    list->dummyHead->next = list->dummyTail;

    list->dummyTail->prev = list->dummyHead;

    list->dummyTail->next = NULL;

    list->size = 0;

}

*/\* 判断链表是否为空 \*/*

bool isEmpty(SignalList \*list) {

    return list->size == 0;

}

*/\* 创建新节点 \*/*

Node\* createNode(ElemType x, ElemType y, ElemType z) {

    Node \*node = (Node\*)malloc(sizeof(Node));

    node->x = x;

    node->y = y;

    node->z = z;

    node->prev = NULL;

    node->next = NULL;

    return node;

}

*/\* 在指定位置插入节点 \*/*

void insert(SignalList \*list, int index, ElemType x, ElemType y, ElemType z) {

    if(index < 0 || index > list->size) return;

    Node \*prev = list->dummyHead;

    for(int i=0; i<index; i++)

        prev = prev->next;

    Node \*newNode = createNode(x, y, z);

*// 双向指针连接*

    newNode->next = prev->next;

    newNode->prev = prev;

    prev->next->prev = newNode;

    prev->next = newNode;

    list->size++;

}

*/\* 删除指定位置节点 \*/*

void removeAt(SignalList \*list, int index) {

    if(index <0 || index >= list->size) return;

    Node \*prev = list->dummyHead;

    for(int i=0; i<index; i++)

        prev = prev->next;

    Node \*toDelete = prev->next;

*// 调整链表连接*

    prev->next = toDelete->next;

    toDelete->next->prev = prev;

    free(toDelete);

    list->size--;

}

*/\* 获取节点数据 \*/*

void get(SignalList \*list, int index, ElemType \*x, ElemType \*y, ElemType \*z) {

    if(index <0 || index >= list->size) return;

    Node \*current = list->dummyHead->next;

    for(int i=0; i<index; i++)

        current = current->next;

    \*x = current->x;

    \*y = current->y;

    \*z = current->z;

}

*/\* 均值滤波（窗口大小3）\*/*

void smooth(SignalList \*list) {

    if(list->size < 3) return;

    Node \*current = list->dummyHead->next->next; *// 从第二个节点开始*

*//边界条件：跳过第一个和最后一个*

    for(int i=1; i<list->size-1; i++) {

        Node \*prev = current->prev;

        Node \*next = current->next;

*// 三维坐标分别求平均*

        current->x = (prev->x + current->x + next->x)/3;

        current->y = (prev->y + current->y + next->y)/3;

        current->z = (prev->z + current->z + next->z)/3;

        current = current->next;

    }

}

*/\* 上采样（线性插值）\*/*

void upsample(SignalList \*list) {

    Node \*current = list->dummyHead->next;

    while(current != list->dummyTail && current->next != list->dummyTail) {

        Node \*next = current->next;

*// 在两个节点间插入插值节点*

        insert(list, current->next->prev->x+1,

              (current->x + next->x)/2,

              (current->y + next->y)/2,

              (current->z + next->z)/2);

        current = next;

    }

}

*/\* 下采样（隔点采样）\*/*

void downsample(SignalList \*list) {

    int count = 0;

    Node \*current = list->dummyHead->next;

    while(current != list->dummyTail) {

        Node \*next = current->next;

        if(count % 2 == 1) {

*// 删除奇数位置节点*

            current->prev->next = current->next;

            current->next->prev = current->prev;

            free(current);

            list->size--;

        }

        count++;

        current = next;

    }

}

*/\* 打印信号 \*/*

void printList(SignalList \*list) {

    Node \*current = list->dummyHead->next;

    printf("Signal List (%d points):\n", list->size);

    int index = 1;

    while(current != list->dummyTail) {

        printf("└─ %d\n", index);

        printf("    └─ %.2f\n", current->x);

        printf("    └─ %.2f\n", current->y);

        printf("    └─ %.2f\n", current->z);

        current = current->next;

        index++;

    }

}

*/\* 从文件读取信号数据 \*/*

bool loadFromFile(SignalList \*list, const char \*filename) {

    FILE \*file = fopen(filename, "r");

    if (file == NULL) {

        printf("无法打开文件 %s\n", filename);

        return false;

    }

*// 清空现有列表*

    while (!isEmpty(list)) {

        removeAt(list, 0);

    }

    char line[100];

    int index;

    ElemType x, y, z;

    while (fgets(line, sizeof(line), file)) {

        if (sscanf(line, "└─ %d", &index) == 1) {

*// 读取坐标*

            fgets(line, sizeof(line), file);

            sscanf(line, "    └─ %f", &x);

            fgets(line, sizeof(line), file);

            sscanf(line, "    └─ %f", &y);

            fgets(line, sizeof(line), file);

            sscanf(line, "    └─ %f", &z);

*// 添加到列表末尾*

            insert(list, list->size, x, y, z);

        }

    }

    fclose(file);

    return true;

}

*/\* 保存信号数据到文件 \*/*

bool saveToFile(SignalList \*list, const char \*filename) {

    FILE \*file = fopen(filename, "w");

    if (file == NULL) {

        printf("无法创建文件 %s\n", filename);

        return false;

    }

    Node \*current = list->dummyHead->next;

    int index = 1;

    while (current != list->dummyTail) {

        fprintf(file, "└─ %d\n", index);

        fprintf(file, "    └─ %.2f\n", current->x);

        fprintf(file, "    └─ %.2f\n", current->y);

        fprintf(file, "    └─ %.2f\n", current->z);

        current = current->next;

        index++;

    }

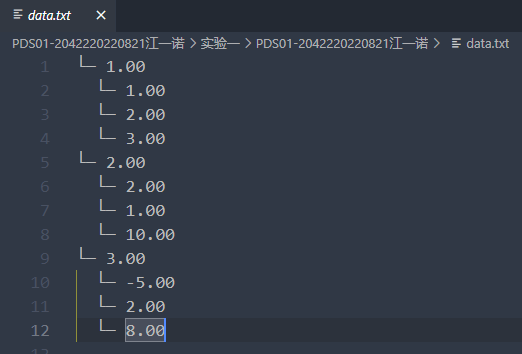
    fclose(file);

    return true;

}

## 5. 实验结果与分析

### 5.1测试用例（初始参数设置）



### 5.2测试程序

int main() {

    SignalList signal;

    init(&signal);

*// 从文件加载信号数据*

    if (loadFromFile(&signal, "data.txt")) {

        printf("\n######从文件加载的原始信号######\n");

        printList(&signal);

*// 均值滤波*

        smooth(&signal);

        printf("\n######滤波后######\n");

        printList(&signal);

*// 保存处理后的数据*

        saveToFile(&signal, "filtered\_signal.txt");

        printf("\n滤波后的数据已保存到 filtered\_signal.txt\n");

*// 上采样*

        upsample(&signal);

        printf("\n##########上采样后##########\n");

        printList(&signal);

*// 保存上采样后的数据*

        saveToFile(&signal, "upsampled\_signal.txt");

        printf("\n上采样后的数据已保存到 upsampled\_signal.txt\n");

*// 下采样*

        downsample(&signal);

        printf("\n##########下采样后###########\n");

        printList(&signal);

*// 保存下采样后的数据*

        saveToFile(&signal, "downsampled\_signal.txt");

        printf("\n下采样后的数据已保存到 downsampled\_signal.txt\n");

    }

*//插入新节点*

    insert(&signal, 3, 1.0, 2.0, 3.0);

    printf("\n########插入新节点后#######\n");

    printList(&signal);

    printf("是否为空:\t");

    isEmpty(&signal)? printf("为空\n") : printf("不为空\n");

*//删除所有节点*

    while (!isEmpty(&signal)) {

        removeAt(&signal, 0);

    }

    printf("\n######清空后的信号######\n");

    printList(&signal);

    printf("是否为空:\t");

    isEmpty(&signal)? printf("为空\n") : printf("不为空\n");

*// 释放资源*

    free(signal.dummyHead);

    free(signal.dummyTail);

    return 0;

}aaaaaaaaaaaaaaaaaaaaaaaaaaaa

### 5.3实验结果

**终端运行结果：**

(base) User@MacBookAir Data structure experiment % cd "/Users/User/Desktop/Data structure experiment/PDS01-2042220220821江一诺

/实验一/PDS01-2042220220821江一诺/" && gcc 实验一.c -o 实验一 && "/Users/User/Desktop/Data structure experiment/PDS01-20422202

20821江一诺/实验一/PDS01-2042220220821江一诺/"实验一

#*#####从文件加载的原始信号######*

Signal List (3 points):

└─ 1

    └─ 1.00

    └─ 2.00

    └─ 3.00

└─ 2

    └─ 2.00

    └─ 1.00

    └─ 10.00

└─ 3

    └─ -5.00

    └─ 2.00

    └─ 8.00

#*#####滤波后######*

Signal List (3 points):

└─ 1

    └─ 1.00

    └─ 2.00

    └─ 3.00

└─ 2

    └─ -0.67

    └─ 1.67

    └─ 7.00

└─ 3

    └─ -5.00

    └─ 2.00

    └─ 8.00

滤波后的数据已保存到 filtered\_signal.txt

#*#########上采样后##########*

Signal List (6 points):

└─ 1

    └─ -0.25

    └─ 1.75

    └─ 6.00

└─ 2

    └─ -2.42

    └─ 1.92

    └─ 6.50

└─ 3

    └─ 1.00

    └─ 2.00

    └─ 3.00

└─ 4

    └─ -0.67

    └─ 1.67

    └─ 7.00

└─ 5

    └─ 0.17

    └─ 1.83

    └─ 5.00

└─ 6

    └─ -5.00

    └─ 2.00

    └─ 8.00

上采样后的数据已保存到 upsampled\_signal.txt

#*#########下采样后###########*

Signal List (3 points):

└─ 1

    └─ -0.25

    └─ 1.75

    └─ 6.00

└─ 2

    └─ 1.00

    └─ 2.00

    └─ 3.00

└─ 3

    └─ 0.17

    └─ 1.83

    └─ 5.00

下采样后的数据已保存到 downsampled\_signal.txt

#*#######插入新节点后#######*

Signal List (4 points):

└─ 1

    └─ -0.25

    └─ 1.75

    └─ 6.00

└─ 2

    └─ 1.00

    └─ 2.00

    └─ 3.00

└─ 3

    └─ 0.17

    └─ 1.83

    └─ 5.00

└─ 4

    └─ 1.00

    └─ 2.00

    └─ 3.00

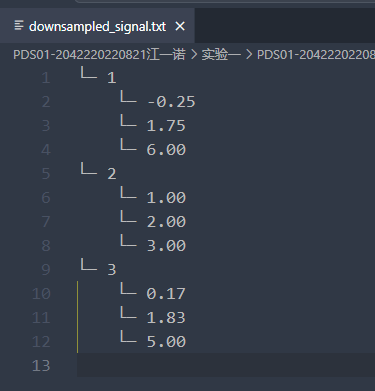
是否为空:       不为空

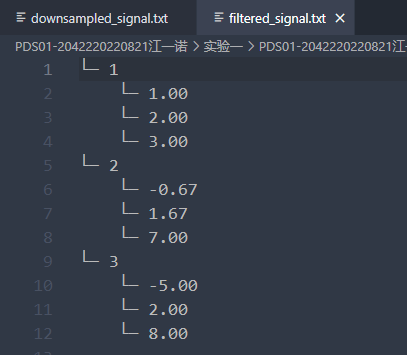
#*#####清空后的信号######*

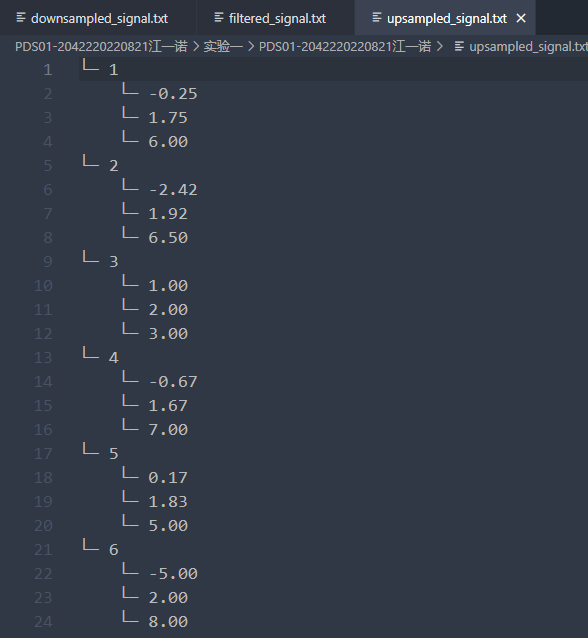
Signal List (0 points):

是否为空:       为空

**保存文件：**







### 5.4实验分析

**(1)时间和空间复杂度分析**

**时间复杂度**

* **初始化(init)**: O(1)
* **插入(insert)**: O(n)，因需遍历到第index个位置
* **删除(removeAt)**: O(n)，同样需要遍历
* **获取节点(get)**: O(n)
* **均值滤波(smooth)**: O(n)
* **上采样(upsample)**: O(n)，但会导致链表规模增加一倍
* **下采样(downsample)**: O(n)

**空间复杂度**

* 整体数据结构: O(n)，n为信号点数量
* 各操作额外空间: O(1)，除上采样每次会增加O(n)空间

**(2)边界条件分析**

1. **空链表处理**:

* isEmpty函数能正确判断
* smooth、upsample等操作对空链表有防护

1. **插入和删除边界**:

* 插入和删除函数对越界index有检查，但返回值缺失

1. **滤波边界问题**:

* smooth函数正确处理了首尾节点不参与计算的边界条件

**(3)实验中出现的问题与改进方案**

* 1. **内存泄漏风险**:
* 程序最后只释放了dummyHead和dummyTail，若操作中出现异常退出，可能导致节点内存未释放
* 应添加销毁链表函数
  1. **错误处理不足**:
* 插入、删除等操作没有返回状态
* 文件操作错误处理简单

**(4)实验结论**

1. **数据结构选择**:

* 双向链表适合信号处理的动态插入删除操作
* 虚拟头尾节点设计简化了边界处理

1. **算法局限性**:

* 均值滤波仅使用固定窗口大小(3)，缺乏可调节性
* 线性插值上采样假设信号连续平滑，对突变信号不适用
* 下采样简单丢弃节点，可能丢失关键信息

1. **优化方向**:

* 基于频率域的采样算法可能更精确
* 对非均匀采样信号的支持不足
* 可添加更多滤波器类型(中值、高通等)