实验报告电子版

**数据结构实训**

**—— 多维数字信号的查找**

姓 名： 刘远明

学 号： 2220212113

指导老师： 韩凤

实验日期： 2024年5月6日

**大连海事大学电子信息科学技术专业**

**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.

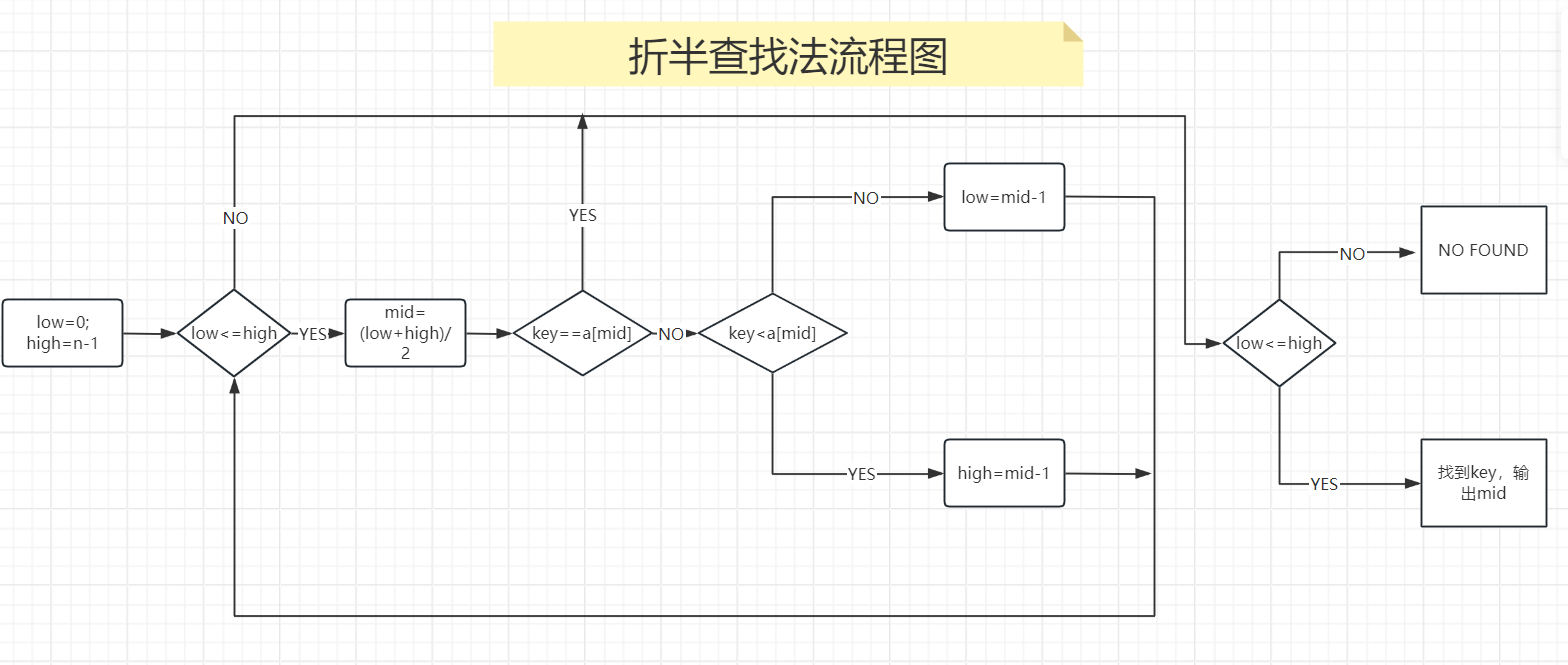
1. **实验目的**
2. 深入理解各种查找的算法思想、方法、稳定性及时间和空间复杂度；
3. 掌握静态表、动态表和哈希查找的算法实现；
4. 能够实现多维信号的查找操作；
5. 提高实际动手进行程序设计的能力。
6. **实验内容与要求**

**实验内容：**

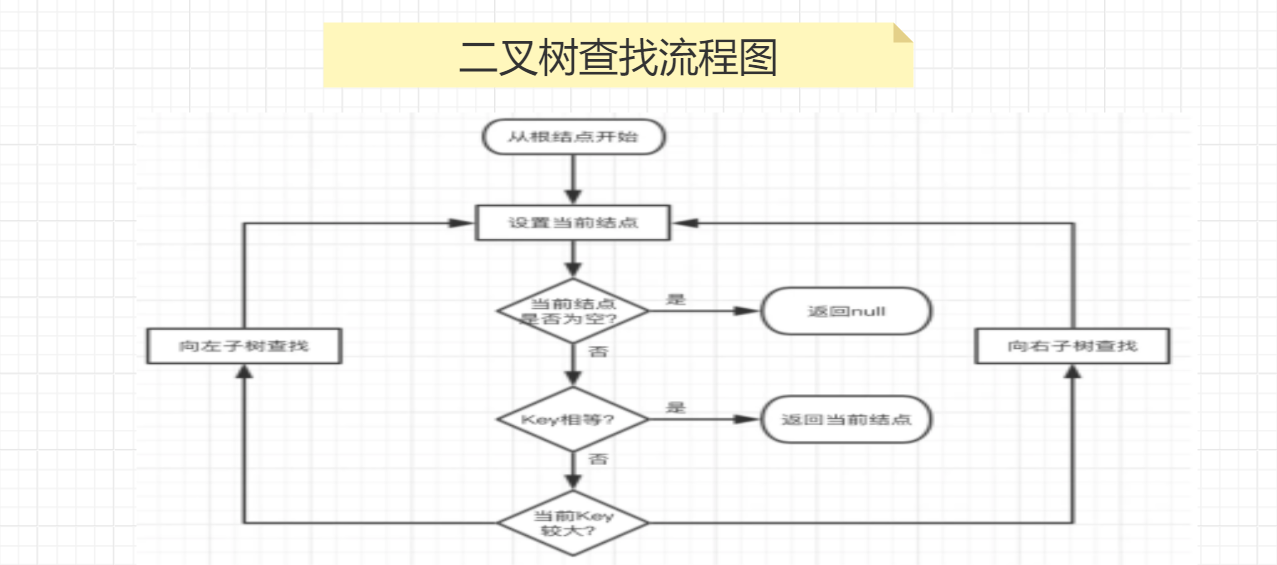
1. 从静态表、动态表和哈希表三类查找算法中各选择一种算法实现；
2. 查找的依据可以是其中的任意一维或者所有维数；
3. 距离的计算方式可以为欧几里得距离、街区距离或者余弦距离；
4. 比较分析三种算法的优缺点和适用范围。

**实验要求：**

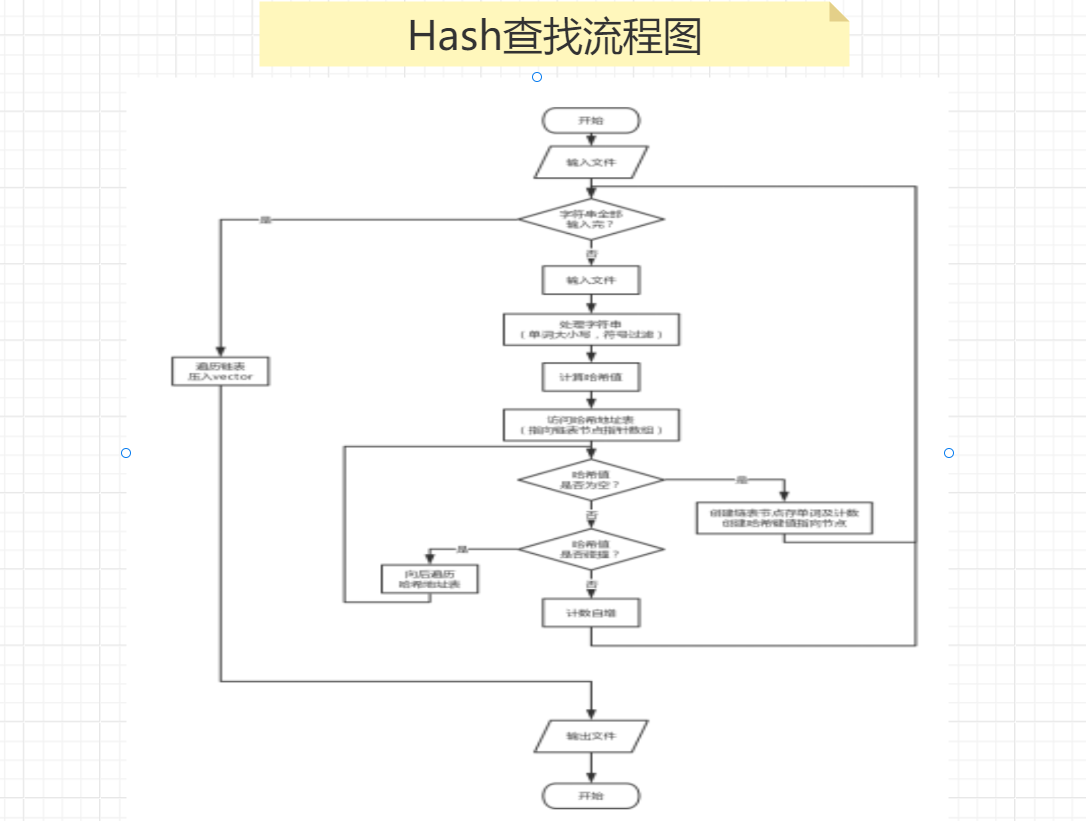
1. 遵守工程伦理和学术规范。
2. 解决的问题要来源于实际工程：要对日常生活中的一个具体应用进行抽象建模与求解，鼓励对国家重大工程中的一个具体点进行抽象建模与求解并融入工程管理理念。
3. 深入理解各种查找算法思想、方法及时间和空间复杂度，要对各种方案进行分析比较。
4. 要对实验结果进行和解释，并通过信息综合得到合理有效地结论。
5. **算法描述与流程**
6. 折半查找法流程图



1. 二叉排序树查找流程图



1. Hash查找流程图



**本次实验使用了折半查找法、二叉排序树法、哈希表查找法对随机生成的数据集进行排序。随机生成的数据集的关键字值使用余弦距离来计算**

1. **算法实现**

**#define \_CRT\_SECURE\_NO\_WARNINGS**

**#include <stdio.h>**

**#include <stdlib.h>**

**#include <string.h>**

**#include <math.h>**

**#include <iostream>**

**#include<time.h>**

**#define MAX\_LENGTH 100**

**#define NIL 0**

**#define MAX\_DATA 1000**

**clock\_t start, finish; //定义第一次调用CPU时钟单位的实际，可以理解为定义一个计数器**

**double Total\_time; //定义一个double类型的变量，用于存储时间单位**

**int k = 0;**

**int sz = 1000;**

**int left = 0;//左下标**

**int right = sz - 1;//右下标**

**char str1[] = "AAAA";**

**int a = 0;**

**typedef struct {**

**char letters[5]; // 4个字母和一个结尾符号**

**int nums[3];**

**int id;**

**} DataEntry;**

**typedef struct {**

**int id0;**

**int dis;**

**} distance;**

**DataEntry data[MAX\_DATA];**

**char line[MAX\_LENGTH];**

**int index = 0;**

**int read\_data() {**

**FILE \*file = fopen("E:\\平时作业\\数据结构实验\\实验五\\data.txt", "r");**

**if (file == NULL) {**

**printf("Error opening file.\n");**

**return 1;**

**}**

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

**sscanf(line, "%4s,%d,%d,%d", data[index].letters, &data[index].nums[0], &data[index].nums[1], &data[index].nums[2]);**

**data[index].id = index;**

**index++;**

**}**

**fclose(file);**

**// 打印读取到的数据**

**for (int i = 0; i < index; i++) {**

**printf("%d,%s,%d,%d,%d\n", data[i].id, data[i].letters, data[i].nums[0], data[i].nums[1], data[i].nums[2]);**

**}**

**}**

**int Division(distance R[], int low, int high) //分割；**

**{**

**distance base = R[low]; //基准元素；**

**while (low < high)**

**{**

**//先从后向前找，再从前向后找**

**while (low < high && R[high].dis >= base.dis)**

**high--; //从右向左找第一个比基准值小的元素；**

**R[low] = R[high];**

**while (low < high && R[low].dis < base.dis)**

**low++; //从左向右找第一个比基准值大的元素；**

**R[high] = R[low];**

**}**

**R[low] = base;**

**return low;**

**}**

**void QuickSort(distance R[], int low, int high)**

**{**

**if (low < high)**

**{**

**int i = Division(R, low, high); //分割；**

**QuickSort(R, low, i - 1); //将两部分分别排序；**

**QuickSort(R, i + 1, high);**

**}**

**}**

**typedef struct SortTree {**

**int data;//存放数据的数据域**

**struct SortTree\* left;//指针域 左指针**

**struct SortTree\* right;//指针域 右指针**

**}Node;**

**/\*全局变量\*/**

**Node\* root;//根节点**

**void Init(int key)**

**{**

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

**root->data = key;**

**root->left = NULL;**

**root->right = NULL;**

**}**

**void insert(int key)**

**{**

**//定义一个临时指针 用于移动**

**Node\* temp = root;//方便移动 以及 跳出循环**

**Node\* prev = NULL;//定位到待插入位置的前一个结点**

**while (temp != NULL)**

**{**

**prev = temp;**

**if (key < temp->data)**

**{**

**temp = temp->left;**

**}**

**else if (key > temp->data)**

**{**

**temp = temp->right;**

**}**

**else**

**{**

**return;**

**}**

**}**

**if (key < prev->data)**

**{**

**prev->left = (Node\*)malloc(sizeof(Node));**

**prev->left->data = key;**

**prev->left->left = NULL;**

**prev->left->right = NULL;**

**}**

**else**

**{**

**prev->right = (Node\*)malloc(sizeof(Node));**

**prev->right->data = key;**

**prev->right->left = NULL;**

**prev->right->right = NULL;**

**}**

**}**

**void show(Node\* root)**

**{**

**if (root == NULL)**

**{**

**return;**

**}**

**show(root->left);**

**printf("%d ", root->data);**

**show(root->right);**

**}**

**int search(Node\* root, int key)**

**{**

**while (root != NULL)**

**{**

**if (key == root->data)**

**return root->data;**

**else if (key < root->data)**

**root = root->left;**

**else**

**root = root->right;**

**}**

**return 0;**

**}**

**typedef struct {**

**int\* Table;//储存哈希节点的数组基地址**

**int size;//哈希表长度**

**}HashTable;**

**//初始化哈希表**

**HashTable\* InitHashTabel(int size) {**

**HashTable\* H = (HashTable\*)malloc(sizeof(HashTable));**

**H->size = size;**

**H->Table = (int\*)malloc(sizeof(int) \* size);**

**//将所以槽位初始化为空闲状态**

**while (size > 0) H->Table[--size] = NIL;**

**return H;**

**}**

**//哈希函数**

**int Hash(int data, int size) {**

**return data % size;//除留余数法**

**}**

**//线性探测法解决哈希冲突**

**int LinearProbe(HashTable\* H, int data) {**

**int Pos = Hash(data, H->size);//通过哈希函数计算得到其哈希地址**

**//若当前位置被占用**

**while (H->Table[Pos] != NIL) {**

**//若已存在当前键**

**if (H->Table[Pos] == data) {**

**return Pos;//返回其位置**

**}**

**Pos = (Pos + 1) % H->size;//线性探测下一个槽位**

**}**

**return Pos;//返回空闲位置**

**}**

**//插入哈希节点**

**int Insert(HashTable\* H, int key) {**

**int Pos = LinearProbe(H, key);//获取该关键字应所在位置**

**//判断该关键字是否在哈希表中**

**if (H->Table[Pos] != key) {**

**H->Table[Pos] = key;**

**return 1;//插入成功**

**}**

**return 0;//插入失败**

**}**

**//查询哈希节点**

**int Search(HashTable\* H, int key) {**

**//线性探测查找key是否在哈希表中**

**int Pos = LinearProbe(H, key);**

**if (H->Table[Pos] != NIL)**

**return Pos;**

**return -1;//所查元素不存在**

**}**

**//删除哈希节点**

**int Delete(HashTable\* H, int key) {**

**int Pos = Search(H, key);//查找该关键字**

**if (Pos != -1) {**

**H->Table[Pos] = NIL;//直接将槽位置空**

**return 1;//删除成功，返回1**

**}**

**return 0;//删除失败，返回0**

**}**

**//打印哈希表节点**

**void print(HashTable\* H) {**

**for (int i = 0; i < H->size; i++) {**

**printf("%d ", H->Table[i]);**

**}**

**}**

**int get\_charnum() {**

**printf("请输入你需要查找的字符\n");**

**scanf("%s", &str1);**

**for (int i = 1; i < 1000; i++) {**

**if (!strcmp(str1, data[i].letters))**

**{**

**k = data[i].id;**

**break;**

**}**

**}**

**return k;**

**}**

**/\*\*/**

**int main()**

**{**

**read\_data();**

**distance distance[1000];**

**for (int i = 1; i < 1000; i++) {**

**distance[i].dis = sqrt(data[i].nums[0] \* data[i].nums[0] + data[i].nums[1] \* data[i].nums[1] + data[i].nums[2] \* data[i].nums[2]);**

**distance[i].id0 = i;**

**}**

**Init(500);**

**for (int i = 1; i < 999; i++)**

**{**

**insert(distance[i].id0);**

**}**

**show(root);**

**HashTable\* H = InitHashTabel(1000);**

**for (int i = 1; i < 1000; i++) {**

**Insert(H, distance[i].id0);**

**}**

**print(H);**

**while (1)**

**{**

**printf("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");**

**printf("\* 1.二分查找 \*\n");**

**printf("\* 2.二叉排序查找 \*\n");**

**printf("\* 3.Hash查找 \*\n");**

**printf("\* 4.退出 \*\n");**

**printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");**

**printf("初始化已完成，请输入操作：\n");**

**int input;**

**scanf("%d", &input);**

**switch (input) {**

**case 1:**

**start = clock();**

**k = get\_charnum();**

**while (left <= right)**

**{**

**int mid = (left + right) / 2;//中间元素下标;mid = left+(right-left)/2**

**if (distance[mid].id0 < k)**

**{**

**left = mid + 1;**

**}**

**else if (distance[mid].id0 > k)**

**{**

**right = mid - 1;**

**}**

**else**

**{**

**printf("找到了下标是：%d\n", mid);**

**break;**

**}**

**if (left > right)**

**printf("不存在该数字\n");**

**}**

**finish = clock();**

**Total\_time = finish - start;**

**break;**

**case 2: {**

**start = clock();**

**k = get\_charnum();**

**int a = search(root, k);**

**printf("\n%d\n", a);**

**finish = clock();**

**Total\_time = finish - start;**

**break;**

**}**

**case 3:**

**start = clock();**

**k = get\_charnum();**

**a = Search(H, k);**

**printf("\n%d\n", a);**

**finish = clock();**

**Total\_time = finish - start;**

**break;**

**case 4:**

**exit(0);**

**break;**

**default:**

**printf("输入操作项有误，请重新输入\n");**

**break;**

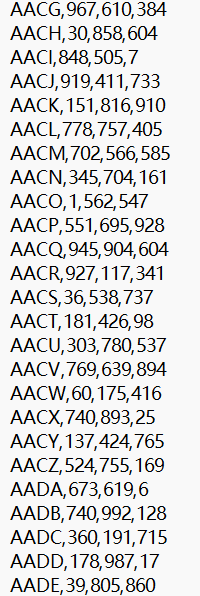
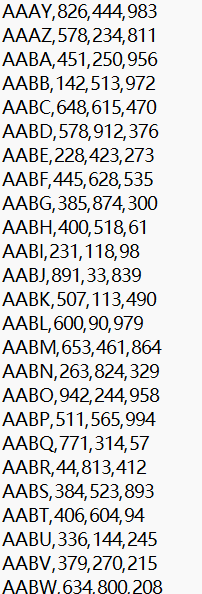
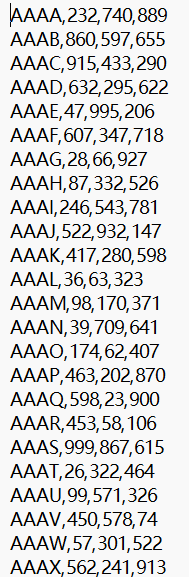
**}**

**}**

**}**

1. **实验结果与分析**

**5.1测试用例**



**5.2测试程序**

**int main()**

**{**

**read\_data();**

**distance distance[1000];**

**for (int i = 1; i < 1000; i++) {**

**distance[i].dis = sqrt(data[i].nums[0] \* data[i].nums[0] + data[i].nums[1] \* data[i].nums[1] + data[i].nums[2] \* data[i].nums[2]);**

**distance[i].id0 = i;**

**}**

**Init(500);**

**for (int i = 1; i < 999; i++)**

**{**

**insert(distance[i].id0);**

**}**

**show(root);**

**HashTable\* H = InitHashTabel(1000);**

**for (int i = 1; i < 1000; i++) {**

**Insert(H, distance[i].id0);**

**}**

**print(H);**

**while (1)**

**{**

**printf("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");**

**printf("\* 1.二分查找 \*\n");**

**printf("\* 2.二叉排序查找 \*\n");**

**printf("\* 3.Hash查找 \*\n");**

**printf("\* 4.退出 \*\n");**

**printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");**

**printf("初始化已完成，请输入操作：\n");**

**int input;**

**scanf("%d", &input);**

**switch (input) {**

**case 1:**

**start = clock();**

**k = get\_charnum();**

**while (left <= right)**

**{**

**int mid = (left + right) / 2;//中间元素下标;mid = left+(right-left)/2**

**if (distance[mid].id0 < k)**

**{**

**left = mid + 1;**

**}**

**else if (distance[mid].id0 > k)**

**{**

**right = mid - 1;**

**}**

**else**

**{**

**printf("找到了下标是：%d\n", mid);**

**break;**

**}**

**if (left > right)**

**printf("不存在该数字\n");**

**}**

**finish = clock();**

**Total\_time = finish - start;**

**break;**

**case 2: {**

**start = clock();**

**k = get\_charnum();**

**int a = search(root, k);**

**printf("\n%d\n", a);**

**finish = clock();**

**Total\_time = finish - start;**

**break;**

**}**

**case 3:**

**start = clock();**

**k = get\_charnum();**

**a = Search(H, k);**

**printf("\n%d\n", a);**

**finish = clock();**

**Total\_time = finish - start;**

**break;**

**case 4:**

**exit(0);**

**break;**

**default:**

**printf("输入操作项有误，请重新输入\n");**

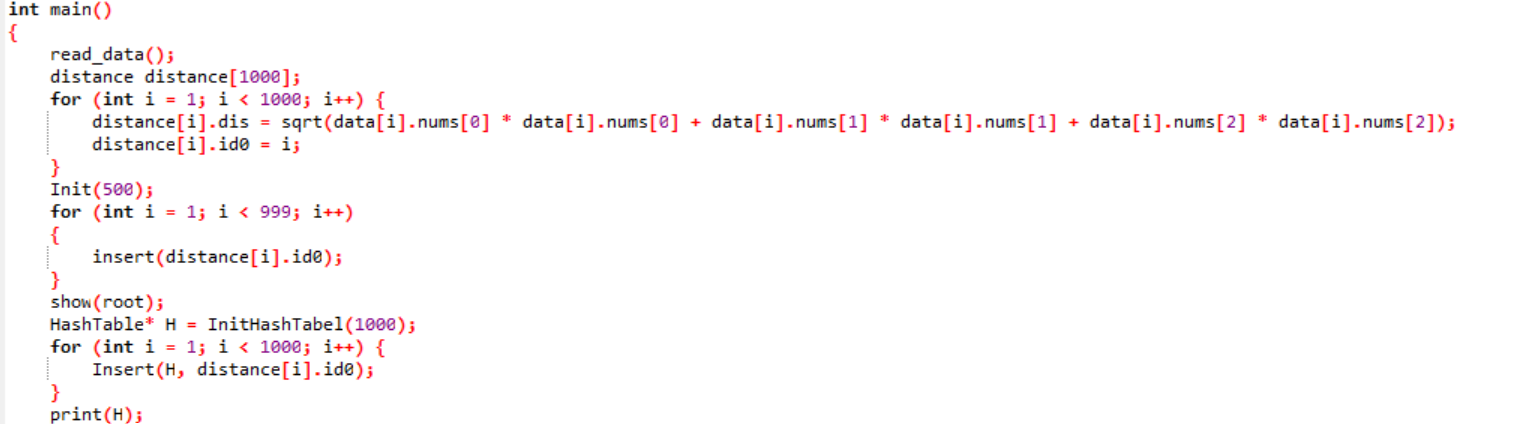
**break;**

**}**

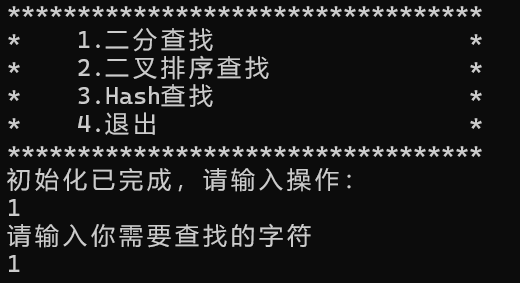
**}**

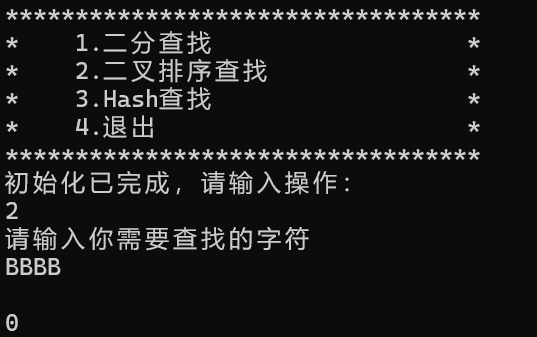
**}**

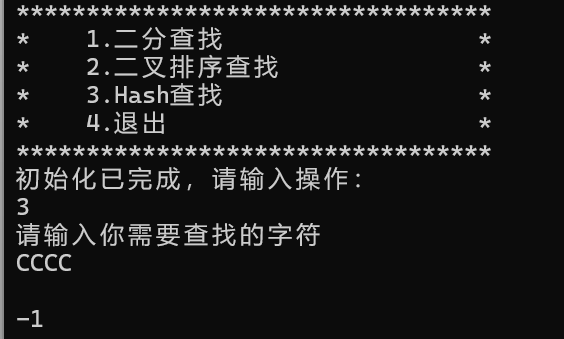
**5.3初始参数设置**

****

**5.4实验结果**

****





**5.5实验分析**

1. **时间复杂度：**

①线性搜索：在多维空间中，线性搜索需要检查每一个点，时间复杂度通常是O(n^d)，其中n是信号点的数量，d是维度数。

②二分搜索：只适用于有序信号，且维度较低时可能有效。时间复杂度为O(log n)，但在多维空间中通常不适用。

③树形搜索：如KD树，其搜索效率依赖于树的平衡性。在平衡情况下，平均时间复杂度接近O(log n)，但在最坏情况下可能接近O(n)。

④哈希搜索：如果能够将多维信号映射到哈希表中，则搜索时间复杂度接近O(1)，但哈希函数的设计和冲突解决策略是关键。

1. **空间复杂度：**

①线性搜索：除了输入数据本身，不需要额外的空间。

②树形搜索：需要额外的空间来存储树结构，空间复杂度取决于树的形状和大小。

③哈希搜索：需要额外的空间来存储哈希表，空间复杂度取决于哈希表的大小和冲突解决策略。

1. **边界条件分析：**

①输入数据的有效性：确保输入信号是有效的，没有缺失值或异常值。

②搜索范围的限制：如果搜索有特定的范围限制（如边界框），需要确保搜索算法不会超出这些范围。

③数据结构的完整性：如果使用了树形结构或哈希表，需要确保在插入和删除元素时保持数据结构的完整性。

1. **运行结果错误原因分析：**

①算法实现错误：检查算法的实现是否完全按照预期进行，特别是边界条件和特殊情况的处理。

②数据问题：检查输入数据是否准确、完整，并符合算法的要求。

③环境问题：确保运行环境（如操作系统、编译器、内存等）没有问题，并且满足算法的运行要求。

④性能瓶颈：如果算法在处理大量数据时变慢或出错，可能是由于内存不足、磁盘I/O瓶颈或其他性能问题。

⑤精度问题：在多维空间中，浮点数的精度问题可能导致意外的结果。确保使用合适的数据类型和精度控制策略。