目 录

[实验一：线性结构的表示与实现 2](#_Toc4050212)

[实验二、矩阵基本操作 15](#_Toc4050213)

[实验三：树和二叉树的基本操作 18](#_Toc4050214)

[实验四：查找与排序 23](#_Toc4050215)

说 明

（1）每个实验题目的程序框架均已给出，请在被空白的方框位置，根据题目要求设计算法或补充函数定义，然后对程序进行编译、调试。

（2）实验报告格式

实验名称

实验目的及要求

实验内容

实验小结

# 实验一：线性结构的表示与实现

**实验目的及要求**

1．熟悉线性表的顺序和链式存储结构

2．掌握线性表的基本运算

3．能够利用线性表的基本运算完成线性表的运算

4．熟悉栈和队列的顺序和链式存储结构

5．掌握栈和队列的基本运算

6．能够利用栈和队列的基本运算完成栈和队列应用的运算

**实验原理**

线性表是最基本、最简单、也是最常用的一种数据结构。表中的每个数据元素，除了第一个外，有且只有一个前件，除了最后一个外，有且只有一个后件，这样的有限序列称之为线性表。线性表常用顺序和链式两种存储结构。

栈和队列是两种特殊的线性表。栈的插入和删除运算都只在线性表的一端进行，另一端是封闭的。允许插入和删除的一端称为栈顶。不允许进行插入和删除的一端称为栈底。具有“先进后出”或者“后进先出”的特点。队列是另一种个数的线性表，只允许在一端插入，而在另一端进行删除。允许插入的端称队尾，允许删除的端称为队头。队列具有“先进先出”或“后进后出”的特点。

**实验设备**

实验所用计算机需满足：

（1）Windows XP及以上版本的操作系统。

（2）已安装TURBO C 或Visual C++。

**实验内容（步骤及方法）**

问题1：线性表表示及实现

实验题目: 线性表顺序和链表表示及实现

实验目的：1学会定义线性表顺序存储结构

2 通过对顺序存储结构的分析，掌握线性表动态分配存储及基本操作，熟悉线性表基本操作及函数定义

3 进一步熟悉C语言基本结构，掌握程序头文件，实现文件及主文件关系及各作用

实验内容：建立10个数据元素的线性表L={1，2，3…10}指定在第2位置插入元素25，然后删除第4个位置上的元素，分别显示各步骤结果

1、顺序表基本操作源码

#include<stdio.h>

#include<stdlib.h>

#include<conio.h>

#define MaxSize 100

typedef int DataType;

typedef struct

{

DataType list[MaxSize];

int size;

}SeqList;

void ListInitiate(SeqList \*L)

{ L->size = 0; /\*定义初始数据元素个数\*/

}

int creat(SeqList \*L)

{int i;

printf("input the datas:");

for(i=0;i<10;i++)

scanf("%d",&L->list[i]);

return 1;

}

int ListLength(SeqList L)

{ return L.size;

}

int ListInsert(SeqList \*L, int i, DataType x)

{ int j;

if(i<0||i>L->size)

{

printf("i is error");

return 0;

}

for(j = L->size; j > i-1; j--)

L->list[j] = L->list[j-1]; /\*依次后移\*/

L->list[i] = x; /\*插入x\*/

L->size ++; /\*元素个数加1\*/

return 1;

}

int ListDelete(SeqList \*L, int i, DataType \*x)

{ int j;

if(L->size<=0)

{printf("顺序表已空，无法删除");

return 0;

}

else

\*x = L->list[i]; /\*保存删除的元素到x中\*/

for(j = i +1; j <=L->size-1; j++)

L->list[j-1] = L->list[j]; /\*依次前移\*/

L->size--; /\*数据元素个数减1\*/

return 1;

}

int ListGet(SeqList L, int i, DataType \*x)

{ if(i < 0 || i > 10)

{

printf("参数i不合法! \n");

return 0;

}

else

{

\*x = L.list[i];

return 1;

}

}

void main()

{

SeqList L;

ListInitiate(&L);

DataType a, b;

creat(&L);

L.size = 10;

printf("初始表：");

for (int i = 0; i < 10; i++)

{

ListGet(L, i, &a);

printf("%d ", a);

}

printf("\n");

ListInsert(&L, 2, 25);

printf("插入后：");

for (int i = 0; i < 11; i++)

{

ListGet(L, i, &a);

printf("%d ", a);

}

printf("\n");

ListDelete(&L, 4, &b);

printf("删除后：");

for (int i = 0; i < 10; i++)

{

ListGet(L, i, &a);

printf("%d ", a);

}

**}**

2、单链表基本操作部分源码

#include<stdio.h>

#include<stdlib.h>

typedef int ElemType;

typedef struct Node

{

ElemType data;

struct Node \*next;

} LNode,\*LinkList;

void inilialist(LNode \*\*L)

{

if((\*L=(LNode \*)malloc(sizeof(LNode)))==NULL)exit(1);

(\*L)->next=NULL;

}

void insert(LNode \*L,int i,ElemType x)

{

LNode \*s,\*p=L;

int j=0;

while(j!=(i-1)&&p!=NULL)

{p=p->next;

j++;

}

if(p==NULL)

{printf("序号出错！");

}

else

{if((s=(LNode\*)malloc(sizeof(LNode)))==NULL) exit(1);

s->data=x;

s->next=p->next;

p->next=s;

}

}

void Delete(LNode \*L,int i)//删除值为x的结点

{

LNode \*p;

LNode \*u;

int j=0;

p=L;

while(j!=i-1&&p!=NULL)

{p=p->next;

j++;}

if(p==NULL||p->next==NULL)

{printf("删除序号出错！");

}

else

{u=p->next;

p->next=u->next;

free(u);

}

}

void main()**{**

LNode \*L;

inilialist(&L);

int i, x;

LNode \*head = L;

for (i = 1; i < 11; i++)

{

scanf("%d", &x);

insert(L, i, x);

}

printf("初始表：");

L = head->next;

while (L)

{

printf("%d ", L->data);

L = L->next;

}

printf("\n");

L = head;

insert(L, 2, 25);

printf("插入后：");

L = head->next;

while (L)

{

printf("%d ", L->data);

L = L->next;

}

printf("\n");

L = head;

Delete(L, 4);

printf("删除后：");

L = head->next;

while (L)

{

printf("%d ", L->data);

L = L->next;

}

**}**

问题2．顺序栈的实践

实验目的：熟练掌握栈的结构、特点及有关概念，学会建立栈的基本操作，了解栈满和栈空的判断条件及描述方法

A、实验内容：建立一个字符栈

⑴用顺序栈设计实现堆栈,堆栈操作集合包括初始化,判断栈空,入栈,出栈,取栈顶元素等

(2)设计一个主函数对顺序栈进行测试,如:依次输入数据元素1,2,3,4,5,然后判断栈是否为空,显示栈顶元素,出栈并在屏幕上显示出栈的数据元素

判字符串中心对称的程序program1\_3代码

#include<stdio.h>

#include<malloc.h>

#include <stdio.h> /\*该文件包含pringtf()等函数\*/

#include <stdlib.h>

#define MaxStackSize 100 /\*定义MaxSize为100\*/

typedef int DataType; /\*定义DataType为int\*/

typedef struct

{

DataType stack[MaxStackSize];

int top;

} SeqStack;

void StackInitiate(SeqStack \*S) /\*初始化顺序堆栈S\*/

{

S->top = -1; /\*定义初始栈顶下标值\*/

}

int StackNotEmpty(SeqStack S)

/\*判顺序堆栈S非空否，非空则返回1，否则返回0\*/

{

if(S.top != -1) return 1;

else return 0;

}

int StackPush(SeqStack \*S, DataType x)

/\*把数据元素值x压入顺序堆栈S，入栈成功则返回1，否则返回0 \*/

{

if(S->top == MaxStackSize-1)

{

printf("堆栈已满无法插入! \n");

return 0;

}

else

{

S->top ++;

S->stack[S->top] = x;

return 1;

}

}

int StackPop(SeqStack \*S, DataType \*d)

/\*弹出顺序堆栈S的栈顶数据元素值到参数d ，出栈成功则返回1，否则返回0\*/

{

if(S->top == -1)

{

printf("堆栈已空无数据元素出栈! \n");

return 0;

}

else

{

\*d = S->stack[S->top];

S->top --;

return 1;

}

}

int StackTop(SeqStack S, DataType \*d)

/\*取顺序堆栈S的当前栈顶数据元素值到参数d ，成功则返回1，否则返回0\*/

{

if(S.top == -1)

{

printf("堆栈已空! \n");

return 0;

}

else

{

\*d = S.stack[S.top];

return 1;

}

}

void main(void) {

SeqStack S;

StackInitiate(&S);

int x, i, n, d;

int a;

for (i = 0; i < 5; i++)

{

scanf("%d", &x);

StackPush(&S, x);

}

n = StackNotEmpty(S);

if (n = 1)

{

printf("栈非空\n");

StackTop(S, &d);

printf("栈顶元素：%d\n", d);

}

else

printf("栈为空\n");

printf("出栈：");

for (i = 0; i < 5; i++)

{

StackPop(&S, &d);

printf("%d ", d);

}

**}**

问题3、栈的综合运用

请利用所学栈的知识设计一个进制转换器，实现十进制到八进制的转换

include<stdio.h>

#define MAX 100

typedef int DataType;

typedef struct

{DataType stack[MAX];

int top;

}seqstack;

void StackInitiate(seqstack \*S) /\*初始化顺序堆栈S\*/

{

S->top = 0; /\*定义初始栈顶下标值\*/

}

int StackNotEmpty(seqstack S)

/\*判顺序堆栈S非空否，非空则返回1，否则返回0\*/

{

if(S.top <= 0) return 0;

else return 1;

}

int StackPush(seqstack \*S, DataType x)

/\*把数据元素值x压入顺序堆栈S，入栈成功则返回1，否则返回0 \*/

{

if(S->top >= MAX)

{

printf("堆栈已满无法插入! \n");

return 0;

}

else

{

S->stack[S->top] = x;

S->top ++;

return 1;

}

}

int StackPop(seqstack \*S, DataType \*d)

/\*弹出顺序堆栈S的栈顶数据元素值到参数d ，出栈成功则返回1，否则返回0\*/

{

if(S->top <= 0)

{

printf("堆栈已空无数据元素出栈! \n");

return 0;

}

else

{

S->top --;

\*d = S->stack[S->top];

return 1;

}

}

int StackTop(seqstack S, DataType \*d)

/\*取顺序堆栈S的当前栈顶数据元素值到参数d ，成功则返回1，否则返回0\*/

{

if(S.top <= 0)

{

printf("堆栈已空! \n");

return 0;

}

else

{

\*d = S.stack[S.top - 1];

return 1;

}

}

void conversion(int i)

{

seqstack S;

int x = 0;

StackInitiate(&S);

while (i > 0)

{

x = i % 8;

StackPush(&S, x);

i /= 8;

}

while (StackNotEmpty(S)) //输出

{

StackPop(&S, &x);

printf("%d", x);

}

}

问题4．回文是指一个字符串从头到尾和从尾到头都一样，如串“abcba””，

设字符串从键盘输入设备一个一个字符读入，请用顺序栈和顺序队列两种算法判断一个字符串是否为回文（提示：由于栈遵循“先进先出”规则，而队列遵循“先入后出”规则，可以将字符串分别入栈和入队列，再行比较两者出栈和出队列的元素是否相同，做出判断）。

循环队列入队出队的程序program1\_4代码

#include <stdio.h>

#include <stdio.h>

#include <string.h>

#define MAXLEN 100

#define MaxStackSize 100

typedef char elementType; //定义具体应用的数据类型elementType

typedef struct

{

elementType data[MAXLEN];

int rear; /\*队尾指针\*/

int front; /\*队头指针\*/

} SeqQueue;

typedef struct

{elementType stack[MaxStackSize];

int top;

} SeqStack;

void dataInitiate(SeqQueue \*Q) /\*初始化顺序循环队列Q\*/

{

Q->rear = 0; /\*定义初始队尾指针下标值\*/

Q->front = 0; /\*定义初始队头指针下标值\*/

}

int queueNotEmpty(SeqQueue Q)

/\*判顺序循环队列Q非空否，非空则返回1，否则返回0\*/

{

if(Q.front == Q.rear) return 0;

else return 1;

}

int enQueue(SeqQueue \*Q,elementType x)

/\*把数据元素值x插入顺序循环队列Q的队尾，成功返回1，失败返回0 \*/

{

if ((Q->rear+1)%MAXLEN == Q->front)

{

return 0;

}

else

{

Q->data[Q->rear] = x;

Q->rear = (Q->rear + 1) % MAXLEN;

return 1;

}

}

int outQueue (SeqQueue \*Q,elementType \*d)

/\*删除顺序循环队列Q的队头元素并赋给d ，成功返回1，失败返回0\*/

{

if (Q->front==Q->rear)

{

return 0;

}

else

{

\*d = Q->data[Q->front];

Q->front = (Q->front + 1) % MAXLEN;

return 1;

}

}

void StackInitiate(SeqStack \*S) /\*初始化顺序堆栈S\*/

{

S->top = 0; /\*定义初始栈顶下标值\*/

}

int StackNotEmpty(SeqStack S)

/\*判顺序堆栈S非空否，非空则返回1，否则返回0\*/

{

if(S.top <= 0) return 0;

else return 1;

}

int StackPush(SeqStack \*S,elementType x)

/\*把数据元素值x压入顺序堆栈S，入栈成功则返回1，否则返回0 \*/

{

if(S->top >= MaxStackSize)

{

printf("堆栈已满无法插入! \n");

return 0;

}

else

{

S->stack[S->top] = x;

S->top ++;

return 1;

}

}

int StackPop(SeqStack \*S,elementType \*d)

/\*弹出顺序堆栈S的栈顶数据元素值到参数d ，出栈成功则返回1，否则返回0\*/

{

if(S->top <= 0)

{

printf("堆栈已空无数据元素出栈! \n");

return 0;

}

else

{

S->top --;

\*d = S->stack[S->top];

return 1;

}

}

int StackTop(SeqStack S,elementType \*d)

/\*取顺序堆栈S的当前栈顶数据元素值到参数d ，成功则返回1，否则返回0\*/

{

if(S.top <= 0)

{

printf("堆栈已空! \n");

return 0;

}

else

{

\*d = S.stack[S.top - 1];

return 1;

}

}

int main()

{ SeqQueue mydata;

SeqStack myStack;

char x, y;

int i,length;

char str[20];

dataInitiate(&mydata);

StackInitiate(&myStack);

printf("输入一个字符串：");

scanf("%s",str);

length=strlen(str);

for(i=0;i<length;i++)

{

enQueue (&mydata, str[i]);

StackPush(&myStack, str[i]);

}

while(queueNotEmpty(mydata) == 1 && StackNotEmpty(myStack) == 1)

{

if(outQueue (&mydata, &x) == 1

&& StackPop(&myStack, &y) == 1 && x != y)

{

printf("%s不是回文!\n", str);

break;

}

}

if(queueNotEmpty(mydata)==0 || StackNotEmpty(myStack)==0)

printf("%s是回文!\n", str);

}

**实验注意事项**

（1）问题1中注意，要求逆线性表占用原线性表空间，在设计算法时不要另开辟等大小的线性表空间。

（2）涉及链表操作时，一定要注意，指针要指向明确的空间，杜绝指针任意指向。

**思考讨论题**

（1）栈空的判断条件是什么？

（2）队满与队空的判断条件分别是什么？

（3）入栈、出栈、入队、出队时，对数据的操作与指针的移动孰先孰后？

（4）当问题1中的链表存储方式如果采用不带头结点的单链表存储方式，那么如何利用O(1)的空间复杂度实现对链表的原地逆转。

（5）已知由不具有头结点的单链表表示的线性表中，含有三类字符的数据元素（字母、数字和其他字符），试编写算法构造三个以链表表示的线性表，使每个表中只含有同一类的字符，且利用原表中的结点空间，头结点可另辟空间。

（6）已知存在两个有序的线性链表，试编写算法将其合并为一个有序线性链表。

# 实验二、矩阵基本操作

**实验目的及要求**

1、掌握稀疏矩阵的三元组顺序表、带行逻辑链接的三元组、十字链表这三种存储结构的实现。

2、掌握稀疏矩阵的加法、转置、乘法等基本运算。

3、通过对稀疏矩阵基本运算的实现加深对线性表的顺序和链式存储的理解。

**实验原理**

矩阵转置算法有两种：

1.按列 col(1≤col≤a.nu)扫描三元表a.data，找出所有列号等于col的那些三元组，将它们的行号和列号互换后依次放入b.data中 －**跳着找，顺着存**

2. 先计算出A中每列第一元素在B的三元表中的存储位置（用pot[]数组保存），然后按行扫描A的三元表的元素－**顺着找，跳着存**

**实验设备**

实验所用计算机需满足：

（1）Windows XP及以上版本的操作系统。

（2）已安装TURBO C 或Visual C++。

**实验内容（步骤及方法）**

输入一个稀疏矩阵A,建立A的三元组顺序表,求A的转置矩阵B,请用两种转置法实现稀疏矩阵的转置,注意矩阵的建立、输出的实现。

#include <stdio.h>

#include<malloc.h>

#include <string.h>

typedef int Telemtype;

#define Ok 1

#define Maxsize 10

typedef struct Triple{ /\*定义三元组表\*/

int i,j;

Telemtype v;

}Triple;

typedef struct TSMatrix{

Triple data[Maxsize];

int m;

int n;

int t;

}TSMatrix;

void InitTriple (TSMatrix \*a){ /\*输入三元组表\*/

int i,j,k,val,maxrow,maxcol;

// char contiue;

maxrow=1;

maxcol=1;

i=j=1;

k=0;

while(i!=-1&&j!=-1&&val!=-1){ /\*rol=-1&&col=-1结束输入\*/

printf("input row \n");

scanf("%d",&i);

printf("input col \n");

scanf("%d",&j);

printf("input value\n");

scanf("%d",&val);

a->data[k].i=i;

a->data[k].j=j;

a->data[k].v=val;

if (maxrow<i) maxrow=i;

if (maxcol<j) maxcol=j;

k++;

}

a->m=maxrow;a->n=maxcol;a->t=k-1;

}

void FastTranspose(TSMatrix a, TSMatrix \*b)

{

int p, q, col;

int num[a.n + 1];

int cpot[a.n + 1];

b->m = a.n;

b->n = a.m;

b->t = a.t;

if (a.t != 0)

{

for (col = 1; col <= a.n; col++)

num[col] = 0;

for (int t = 1; t <= a.t; t++)

++num[a.data[t].j];

cpot[1] = 1;

for (col = 2; col <= a.n; col++)

cpot[col] = cpot[col - 1] + num[col - 1];

for (p = 0; p < a.t; p++)

{

col = a.data[p].j;

q = cpot[col]; //p 代表 矩阵a的位置，q代表在b中要插入的位置。

b->data[q].j = a.data[p].i;

b->data[q].i = a.data[p].j;

b->data[q].v = a.data[p].v;

++cpot[col]; //位置发生变化 自动加一

}

}

}

void showMatrix(TSMatrix \*a){ /\*输出稀疏矩阵\*/

int p,q;

int t=0;

for(p=1;p<=a->m;p++)

{for(q=1;q<=a->n;q++)

{ if (a->data[t].i==p&&a->data[t].j==q)

{printf("%d ",a->data[t].v);

t++;

}

else printf("0 ");

}

printf("\n" );

}

}

void TransposeSMatrix(TSMatrix \*a,TSMatrix \*b)

{

int p, q, col;

b->m = a->n;

b->n = a->m;

b->t = a->t;

if (b->t)

{

q = 0;

for (col = 1; col <= a->n; col++)

for (p = 0; p < a->t; p++)

if (a->data[p].j == col)

{

b->data[q].j = a->data[p].i;

b->data[q].i = a->data[p].j;

b->data[q].v = a->data[p].v;

q++;

}

}

}

int main()

{TSMatrix a,b;

InitTriple (&a);

printf("原来矩阵是\n");

showMatrix(&a); /\*转置前\*/

TransposeSMatrix(&a,&b);

printf("普通转置后矩阵是\n");

showMatrix(&b); /\*转置后\*/

FastTranspose(a,&b);

printf("快速转置后矩阵是\n");

showMatrix(&b); /\*转置后\*/

}

# 实验三：树和二叉树的基本操作

**实验目的及要求**

1．熟悉二叉树的链式存储结构

2．掌握二叉树的建立、深度优先递归遍历等算法

3．能够利用遍历算法实现一些应用

**实验原理**

（1）树是一种简单的非线性结构，所有数据原属之间的关系具有明显的层次特性。

（2）二叉树的特点：非空二叉树只有一个根结点；每个结点最多有两个子树，分别称为左子树与右子树。

（3）除最后一层外，每一层上的所有结点都有两个子结点的二叉树为满二叉树。除最后一层外，每一层上的结点数均达到最大值，在最后一层上只缺少右边的若干结点，这样的二叉树为完全二叉树。

（4）二叉树的遍历指不重复地访问二叉树中的所有结点。二叉树的遍历可分为三种，前序遍历、中序遍历和后续遍历，其遍历顺序分别为根左右、左根右、左右根。

**实验设备**

实验所用计算机需满足：

（1）Windows XP及以上版本的操作系统。

（2）已安装TURBO C 或Visual C++。

**实验内容（步骤及方法）**

问题1．二叉树采用二叉链表作存储结构，试编程实现二叉树的如下基本操作：

1. 按先序序列构造一棵二叉链表表示的二叉树T；

2. 对这棵二叉树进行遍历：先序、中序、后序以及层次遍历序列，分别输出结点的遍历序列；

3. 求二叉树的深度/结点数目/叶结点数目；

二叉树操作的program2\_1代码

#include<stdio.h>

#include<malloc.h>

# define Null 0

typedef char ElemType ;

typedef struct node

{

ElemType data ;

struct node \*lchild ;

struct node \*rchild ;

}BTree,\*pBTree ;

//先序创建树

void CreateBTree(BTree \*\*T)

//此处参数应该用指针的指针，应给它要改变指向二叉树根的那个指针

{

char ch ;

printf("请输入先序二叉树\n");

ch=getchar();

getchar(); //得到回车按那个字符

if(ch =='/') //输入空字符时要打空格

{

(\*T) = NULL ;

return ;

}

else

{

if( !( (\*T) = (BTree\*) malloc(sizeof(BTree)) ) ) return ;

(\*T)->data = ch ;

CreateBTree( &(\*T)->lchild );

CreateBTree( &(\*T)->rchild );

}

}

void BTreePrint(BTree \*Tr,int n)

//逆时针旋转90°打印二叉树，n为缩进层数，初始值为0

{

int i;

if(Tr == NULL) return;

BTreePrint(Tr->rchild,n+1);

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

printf(" ");

if(n >= 0)

{

printf("--");

printf("%c\n",Tr->data);

}

BTreePrint(Tr->lchild,n+1);

}

void pre(BTree \*T)

{

if(T)

{

printf("%c", T->data);

pre(T->lchild);

pre(T->rchild);

}

}

void in(BTree \*T)

{if(T)

{

in(T->lchild);

printf("%c",T->data);

in(T->rchild);

}

}

void post(BTree \*T)

{if(T)

{

post(T->lchild);

post(T->rchild);

printf("%c",T->data);

}

}

int getnodes(BTree \*T)

{int n1=0,n2=0;

if(T==Null)

return 0;

else if(T->lchild==Null&&T->rchild==Null)

return 1;

else

{n1=getnodes(T->lchild);

n2=getnodes(T->rchild);

return (n1+n2+1);}

}

int getlefs2(BTree \*T)

{

static int n=0;

if(T)

{

getlefs2(T->lchild);

getlefs2(T->rchild);

if(T->lchild==Null&&T->rchild==Null)

{n++;

}

}

return n;

}

int getlefes(BTree \*T)

{int n1=0,n2=0;

if(T==Null)

return 0;

else if(T->lchild==Null&&T->rchild==Null)

return 1;

else

{n1=getlefes(T->lchild);

n2=getlefes(T->rchild);

return (n1+n2);}

}

int depth(BTree \*T)

{

int h1=0, h2=0;

if (T != Null)

{

h1 = depth(T->lchild);

h2 = depth(T->rchild);

if(h1>h2)

return h1 + 1;

else

return h2 + 1;

}

return 0;

}

int count(BTree \*t)

{static int c=0;

if(t!=Null)

{

count(t->lchild);

c++;

count(t->rchild);

}

return c;}

void main()

{int a,b;

BTree \*bTree ;

CreateBTree(&bTree);

BTreePrint(bTree,0);

printf("先序遍历序列");

pre(bTree);

printf("中序遍历序列");

in(bTree);

printf("后序遍历序列");

printf("\n");

post(bTree);

printf("结点总数是%d\n",getnodes(bTree));

printf("叶子总数是%d\n",getlefes(bTree));

a=count(bTree);

printf("树上结点数是%d\n",a);

b=getlefs2(bTree);

printf("树上叶子%d\n",b);

printf("树的高度是%d\n",depth(bTree));

}

**实验注意事项**

（1）遍历二叉树时注意递归调用的终结条件。

（2）避免在递归中定义较多变量挤占空间。

**思考讨论题**

1. 已知二叉树的前序和中序遍历，如何求出其后序遍历？
2. 试写出二叉树的非递归遍历算法。
3. 试写出求二叉树深度的算法。

# 实验四：查找与排序

**实验目的及要求**

1．熟悉线性表、二叉排序树和散列表的查找

2．能够编写一些查找的算法

3．熟悉各种内部排序算法

4．能够编写程序显示排序过程中各趟排序的结果

5．能够编写一些排序的算法

**实验原理**

（1）有序表的折半查找

设升序线性表的长度为n，被查元素为x，在该线性表的中查找x的方法为：将x与线性表中间项进行比较，若中间项等于x，则表示查找到，查找结束；若中间项小于x，则在线性表后半部以相同的方法查找；若中间项大于x，则在线性表前半部以相同的方法查找。

（2）分块查找

又称索引顺序查找。将长度为n的线性表分成m个子表，各子表长度可以相等也可以不等，但要求后一个子表中的每一个元素均大于前一个子表中的所有元素。线性表本身采用顺序存储结构。再建立一个索引表，索引表中对每个子表的建立一个索引结点，包括数据域和指针域，数据域中存放子表中的最大元素值，指针域表示子表在原顺序表中的位置。

分块查找首先在索引表中用对分查找方式找到被查元素所在的子表，再在原线性表中子表对应的位置用顺序查找法进行具体查找。

（3）哈希查找

哈希查找是一种直接查找技术。根据各元素关键字的值由映像函数确定各元素的存储位置，查找时再根据关键字计算映像函数的值就能够直接找到该位置。对于不同的关键字，如果映像函数的返回值相同，则称为元素发生冲突。使用不同方法处理冲突哈希查找的关键。常见的冲突处理方法有线性哈希、随机哈希和溢出哈希。

哈希查找的操作步骤：

①用给定的哈希函数构造[哈希表](http://baike.baidu.com/item/%E5%93%88%E5%B8%8C%E8%A1%A8)；

②根据选择的冲突处理方法解决地址冲突；

③在哈希表的基础上执行哈希查找。

1. 二叉排序树查找
2. 直接插入排序：将一个记录插入到已排序好的有序表中，从而得到一个新的记录数增1的有序表。即：先将序列的第1个记录看成是一个有序的子序列，然后从第2个记录逐个进行插入，直至整个序列有序为止。要点：设立哨兵，作为临时存储和判断数组边界之用。
3. 希尔排序：先将整个待排序的记录序列分割成为若干子序列分别进行直接插入排序，待整个序列中的记录“基本有序”时，再对全体记录进行依次直接插入排序。
4. 简单选择排序：在要排序的一组数中，选出最小（或者最大）的一个数与第1个位置的数交换；然后在剩下的数当中再找最小（或者最大）的与第2个位置的数交换，依次类推，直到第n-1个元素（倒数第二个数）和第n个元素（最后一个数）比较为止。
5. 堆排序：首先将一个无序序列建成堆，然后将堆顶元素与无序序列最后一个元素做交换，调整堆至稳定。不断重复调整堆与交换的操作，直至无序序列为空。
6. 冒泡排序：在要排序的一组数中，对当前还未排好序的范围内的全部数，自上而下对相邻的两个数依次进行比较和调整，让较大的数往下沉，较小的往上冒。即：每当两相邻的数比较后发现它们的排序与排序要求相反时，就将它们互换。直至剩下的线性表为空，整个序列有序。
7. 快速排序：从线性表中选取一个元素，设为T，然后将线性表后面小于T的元素移动到前面，而前面大于T的元素移动到后面，结果就将线性表分成两部分，T插入到分界处。之后对前后两个子表再按此方法继续操作，直至所有子表为空。
8. 归并排序：将两个或两个以上的有序表合并成一个新的有序表。

（12） 基数排序：从有效数字的最低位开始直到最高位，按每一位有效数字对线性表元素进行重新排列。

**实验设备**

实验所用计算机需满足：

（1）Windows XP及以上版本的操作系统。

（2）已安装TURBO C 或Visual C++。

**实验内容（步骤及方法）**

问题1．对于给定的一组按照从小到大排序的整数，通过普通查找方法和二分查找的方法判断一个数是否在这个数组中。

程序部分源代码如下：

#include<stdio.h>

#include<stdlib.h>

#define MaxSize 100

typedef int KeyType; //整型

typedef struct

{ KeyType key;

} DataType;

typedef struct

{

DataType list[MaxSize];

int size;

}SeqList;

void ListInitiate(SeqList \*L)

{ L->size = 0; /\*定义初始数据元素个数\*/

}

int creat(SeqList \*L,int n)

{int i;

printf("input the datas:");

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

scanf("%d",&L->list[i]);

return 1;

}

int ListLength(SeqList \*L)

{ return L->size;

}

int SeqSearch(SeqList \*s, KeyType k)

//在表s中顺序查找关键字k，若查找成功，则返回该元素在表中的位置，若查找失败，返回-1

{



}

int BinSearch(SeqList \*s , KeyType k)

//在表s中用折半查找法查找关键字k，若查找成功，则函数值为该元素在表中的位置，若查找失败，返回-1。

{



}

int main()

{



}

问题2．任意给出一组关键字（关键字的个数和值在运行时由键盘输入），构造对应的二叉排序树（每次先找到当前关键字的插入位置，然后将其插入），并对其进行中序遍历，输出得到的序列。

#include<stdio.h>

#include<conio.h>

#include<stdlib.h>

typedef int KeyType;

typedef struct

{

KeyType key;

}DataType;

typedef struct BinSTreeNode

{

DataType elem; //elem含有关键字域

struct BinSTreeNode \*lchild;

struct BinSTreeNode \*rchild;

} BinSTreeNode ,\*BinSTree;

void BSTreeInsert (BinSTree \*t , KeyType k)

//在二叉排序树中插入值为elem的元素，\*t指向二叉排序树根结点

{



}

void InOrder(BinSTree t)

{

if(t)

{

InOrder((t)->lchild ); /\* 遍历左子树 \*/

printf("%4d",(t)->elem.key);

InOrder((t)->rchild ); /\* 遍历左子树 \*/

}

}

BinSTree BSTreeSearch(BinSTree t, KeyType k)

{

if (t==NULL)

return NULL;

if (t->elem.key==k)

return (t);

if (t->elem.key>k)

return(BSTreeSearch(t->lchild, k));

else

return(BSTreeSearch(t->rchild, k));

}

int BSTreeDelete (BinSTree \*bt , KeyType k)

// 在二叉排序树中删除关键字为k的结点，\*bt指向二叉排序树的根结点。删除成功返回1，不成功返回0

{

BinSTree f,p,q,s;

p=\*bt;

f=NULL;

while (p&&p->elem.key!=k) /\* 查找关键字为key的结点 \*/

{

f=p; /\* f为指向结点\*p的双亲结点的指针 \*/

if (p->elem.key>k)

p=p->lchild; /\* 搜索左子树 \*/

else

p=p->rchild; /\* 搜索右子树 \*/

}

if (p==NULL) return(0); /\* 找不到待删的结点时返回 \*/

if (p->lchild==NULL) /\* 待删结点的左子树为空 \*/

{

if (f==NULL) /\* 待删结点为根结点 \*/

\*bt=p->rchild;

else if (f->lchild==p) /\* 待删结点是其双亲结点的左结点 \*/

f->lchild=p->rchild ;

else f->rchild=p->rchild; /\* 待删结点是其双亲结点的右结点 \*/

free (p) ;

}

else

{

q=p;

s=p->lchild;

while (s->rchild) /\* 在待删结点的左子树中查找最右下结点 \*/

{

q=s;

s=s->rchild;

}

if (q==p) /\* 将最右下结点的左子树链到待删结点上 \*/

q->lchild=s->lchild;

else

q->rchild=s->lchild;

p->elem.key=s->elem.key;

free (s);

}

return(1);

}

main()

{

BinSTree t=NULL,p;

//KeyType k;

int n,i,c=1,op;

int flag;

KeyType a[20],k;

while(c)

{

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

printf("\t\t\t---二叉排序树---\n");

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

printf("\n\t\t\t\* 1---创建二叉排序树 \*");

printf("\n\t\t\t\* 2---中序遍历二叉排序树 \*");

printf("\n\t\t\t\* 3---查找给定关键字 \*");

printf("\n\t\t\t\* 4---删除给定关键字 \*");

printf("\n\t\t\t\* 0---退 出 \*");

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

printf("\t\t\t请选择菜单号(0--4):");

scanf("%d",&op);

getchar();

switch(op)

{

case 1:

printf("创建二叉排序树\n");

printf("请输入待构造二叉排序数的结点总数：");

scanf("%d",&n);

printf("请输入%d个结点的值：",n);

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

{

scanf("%d",&a[i]);

}

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

BSTreeInsert(&t,a[i]);

break;

case 2:

printf("中序遍历二叉排序树\n");

InOrder(t);

break;

case 3:

printf("查找给定关键字\n");

printf("请输入待查找关键字:");

scanf("%d",&k);

p=BSTreeSearch(t,k);

if(p==NULL)

printf("查找失败\n");

else

printf("%#x,%d",p,(p->elem).key);

break;

case 4:

printf("删除给定关键字\n");

printf("请输入待删除关键字:");

scanf("%d",&k);

flag=BSTreeDelete(&t,k) ;

if(flag==0)

printf("删除失败，表中不存在待删除关键字\n");

else

printf("删除的关键字为：%d",k);

break;

case 0:

c=0;

printf("\n\t\t\t程序结束!\n");

break;

default:

printf("\n\t\t\t\t输入错误!请重新输入!\n");

break;

}

}

}

问题3：任意给出一组关键字（关键字的个数和值在运行时由键盘输入），分别采用直接插入排序，希尔排序，冒泡排序，快速排序，选择排序对其进行排序，并输出每个排序算法得到的序列。

#include<stdio.h>

#include<stdlib.h>

#include<conio.h>

#define MAXSIZE 100 //假定顺序表的最大长度为100

typedef int KeyType; //假定关键字类型为整数类型

typedef struct

{

KeyType key; //关键字项

//OtherType other; // 其他项

}DataType; //数据元素类型

typedef struct

{

DataType r[MAXSIZE+1]; //r[0]闲置或充当哨兵

int length; //顺序表长度

} SqList;

#include<stdio.h>

#include<stdlib.h>

#include<conio.h>

#define MAXSIZE 100 //假定顺序表的最大长度为100

typedef int KeyType; //假定关键字类型为整数类型

typedef struct

{

KeyType key; //关键字项

//OtherType other; // 其他项

}DataType; //数据元素类型

typedef struct

{

DataType r[MAXSIZE+1]; //r[0]闲置或充当哨兵

int length; //顺序表长度

} SqList;

void Creat\_SqList(SqList \*S)

{

int i;

printf("请输入顺序表的长度：");

scanf("%d",&(S->length));

printf("请输入%d个关键字：",S->length);

for(i=1;i<=S->length;i++)

scanf("%d",&(S->r[i].key));

}

void StraightInsertSort(SqList \*S) //直接插入排序//对顺序表s中s->r[1..length]作直接插入排序

{

//your code will be written here

**}**

void ShellInsert(SqList \*s,int gap) //步长为gap的插入排序

{

int i,j ;

for(i=gap+1;i<=s->length;i++)

if (s->r[i].key<s->r[i-gap].key) //小于时，需将r[i]插入有序表

{

s->r[0]=s->r[i]; //为统一算法设置监视哨

for(j=i-gap;j>0&&s->r[0].key< s->r[j].key;j=j-gap)

s->r[j+gap]=s->r[j]; //记录后移

s->r[j+gap]=s->r[0]; //插入到正确位置

}

}

void ShellSort( SqList \*S,int gaps[ ],int t)

{ //按增量序列gaps[0，1…，t-1]对顺序表S作希尔排序

int k;

for( k=0;k<t;k++)

ShellInsert (S,gaps[k]);

}

void BubbleSort(SqList \*S)

//对顺序表S作冒泡排序

{



}

int QuickSort1 (SqList \*S, int low, int high)

{

KeyType pivotkey;

S->r[0]=S->r[low]; //以子表的第一个记录作为轴值记录

pivotkey=S->r[low].key; //取轴值记录关键字

while(low<high) //从表的两端交替地向中间扫描

{

while(low<high && S->r[high].key>=pivotkey)

high--;

if(low<high)

S->r[low++]=S->r[high]; //将比轴值记录小的交换到低端

while(low<high && S->r[low].key<=pivotkey)

low++;

if(low<high)

S->r[high--]=S->r[low];

}

S->r[low]=S->r[0]; //轴值（支点）记录到位

return low; //返回轴值（支点）记录所在位置

}

void QuickSort ( SqList \*S, int low, int high)

{ //对顺序表S中的子序列r[low…high]作快速排序

int pivotloc;

if(low<high)

{ pivotloc=QuickSort1 (S, low, high);

//对小于轴值序列实现递归排序

QuickSort (S,low,pivotloc-1);

//对大于轴值序列实现递归排序

QuickSort (S,pivotloc+1,high);

}

}

void SelectSort(SqList \*S)

{

int i,j,t;

for(i=1;i<S->length;i++) /\* 作S->length-1趟选取 \*/

{

for(j=i+1,t=i;j<=S->length;j++)

{

if (S->r[t].key>S->r[j].key)

t=j; /\* t中存放关键字最小的记录下标 \*/

}

if(t!=i)

{

S->r[0]=S->r[t];

S->r[t]=S->r[i];

S->r[i]=S->r[0];

}

}

}

void Out\_SqList(SqList \*S)

{

int i;

for(i=1;i<=S->length;i++)

{

printf("%4d",S->r[i].key);

}

}

void main()

{

int c=1,op,i;

SqList S;

int gaps[10];

int t;

while(c)

{

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

printf("\t\t\t---排序---\n");

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

printf("\n\t\t\t\* 1---创建待排序序列 \*");

printf("\n\t\t\t\* 2---直接插入排序 \*");

printf("\n\t\t\t\* 3---希尔排序 \*");

printf("\n\t\t\t\* 4---冒泡排序 \*");

printf("\n\t\t\t\* 5---快速排序 \*");

printf("\n\t\t\t\* 6---简单选择排序 \*");

printf("\n\t\t\t\* 7---输出排序后序列 \*");

printf("\n\t\t\t\* 0---退 出 \*");

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

printf("\t\t\t请选择菜单号(0--4):");

scanf("%d",&op);

getchar();

switch(op)

{

case 1:

Creat\_SqList(&S);

break;

case 2:

StraightInsertSort(&S);

break;

case 3:

printf("请输入增量序列的个数：");

scanf("%d",&t);

printf("请输入%d增量的值：\n",t);

for(i=0;i<t;i++)

scanf("%d",&gaps[i]);

ShellSort(&S,gaps,t);

break;

case 4:

BubbleSort(&S);

break;

case 5:

QuickSort (&S, 1, S.length);

break;

case 6:

SelectSort(&S);

break;

case 7:

Out\_SqList(&S);

break;

case 0:

c=0;

printf("\n\t\t\t程序结束!\n");

break;

default:

printf("\n\t\t\t\t输入错误!请重新输入!\n");

break;

}

}

}

**实验注意事项**

（1）各种排序算法时间复杂度

**思考讨论题**

（1）比较排序的最低时间复杂度是nO(logn),那么有哪些不是基于比较的排序算法，它们的适用场景是什么？试实现一种不是基于比较的排序算法。