上海理工大学光电信息与计算机工程学院

**《数据结构》实验报告**

****

**专　　业 智能科学与技术**

**学生姓名　　 高浩琦**

**学　 号 2035060413**

**年　　级 2020级**

**指导教师 曹春萍**

**成 绩：**

**教师签字：**

目录

[实验一 线性表的操作 1](#_Toc105759278)

[实验二 栈和队列的操作 2](#_Toc105759279)

[实验三 二叉树的操作 3](#_Toc105759280)

[实验四 图的操作 4](#_Toc105759281)

[实验五 查找操作 5](#_Toc105759282)

[实验六 排序操作 6](#_Toc105759283)

# 实验一 线性表的操作

1.实验目的

1）熟悉C语言的上机环境，掌握C语言的基本结构。

2）学会定义线性表的顺序和链式存储结构

3）学会顺序表和单链表的初始化、插入、删除、查找和输出操作。

2.实验环境（软件环境）

Visual Studio Code

1. 实验内容（附上源代码及必要注释）

#include<bits/stdc++.h>

#include<iostream>

using namespace **std**;

#define **maxn** 100

typedef int **ElemType**;

typedef struct  *//创建顺序表结构体*

{

**ElemType** a[**maxn**];

    int length;

}**sqlist**;

void **InitList**(**sqlist** &l) *//顺序表初始化*

{

    l.length=0;

}

bool **InsertList**(**sqlist** &l,int i,**ElemType** x) *//线性表中插入元素*

{

        if(i>l.length||i<1) return 0;

        else{

            for(int p=l.length;p>=i;p--){

                l.a[p+1]=l.a[p];

            }

            l.a[i]=x;

            l.length++;

            return 1;

        }

}

bool **DeleteList**(**sqlist** &l,**ElemType** x) *//删除顺序表中指定元素*

{

    bool f=1;

    int length=l.length;

    int i;

    for(i=1;i<=length;i++){

        if(l.a[i]==x){

            for(int j=i+1;j<=length;j++){

                l.a[j-1]=l.a[j];

            }

            l.length--;

            f=0;

        }

    }

    if(f) return 0;

    else return 1;

}

int **FindList**(**sqlist** &l,**ElemType** x) *//查找首先在顺序表中出现的元素*

{

    for(int i=1;i<=l.length;i++)

    {

        if(l.a[i]==x) return i;

    }

    return 0;

}

void **OutputList**(**sqlist** &l)

{

    for(int x=1;x<=l.length;x++)

    {

        cout**<<**l.a[x]**<<**" ";

    }

        cout**<<endl**;

}

int **main**()

{

**sqlist** list;

**InitList**(list);

    for(int x=0;x<=10;x++){

        list.a[x+1]=x;

        list.length++;

    }

**OutputList**(list);

    if(**InsertList**(list,6,6)){

        cout**<<**"insert success"**<<endl**;

**OutputList**(list);

    }else

        cout**<<**"insert error"**<<endl**;

    if(**DeleteList**(list,6)){

        cout**<<**"delete success"**<<endl**;

**OutputList**(list);

    }else cout**<<**"delete error"**<<endl**;

    cout**<<**"find?"**<<endl**;

**ElemType** k;

    cin**>>**k;

    int sta=**FindList**(list,k);

    if(sta!=0){

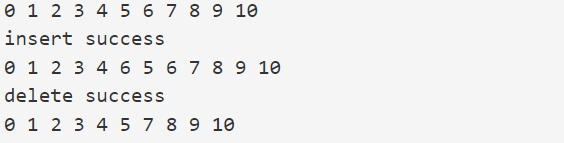
        cout**<<**"位置为第"**<<**sta**<<**"位"**<<endl**;

    }else cout**<<**"未查找到"**<<endl**;

}

1. 实验结果（简单演示说明与截图）

插入和删除



# 实验二 栈和队列的操作

1.实验目的

1）掌握栈和队列的特点。

2）学会定义栈和队列的链式存储结构。

3）掌握入栈和出栈操作

4）掌握入队和出队操作

2.实验环境（软件环境）

Visual Studio Code

1. 实验内容（附上源代码及必要注释）

//栈

#include<cstdlib>

#include<cstring>

#include<iostream>

using namespace **std**;

typedef char **ElemType**;

typedef struct **node**

{

**ElemType** data;

    struct **node** \*next;

}**LinkStack**;

void **initstack**(**LinkStack** \*&ls){

    ls = **NULL**;

}

int **push**(**LinkStack** \*&ls,**ElemType** x){

**LinkStack** \*p;

    p=(**LinkStack** \*)**malloc**(sizeof(**LinkStack**));

    p->data = x;

    p->next = ls;

    ls = p;

**free**(p);

    cout**<<** "success"**<<endl**;

    return  1;

}

void **dispstack**(**LinkStack** \*ls){

    while (ls != **NULL**)

    {

**printf**("%c ",ls->data);

        ls = ls->next;

    }

    cout **<<** **endl**;

}

int **pop**(**LinkStack** \*&ls,**ElemType** x){

**LinkStack** \*p;

    p=(**LinkStack** \*)**malloc**(sizeof(**LinkStack**));

    cout**<<** "success2"**<<endl**;

    if (ls = **NULL**){

        return 0;

    }else{

        p = ls;

        cout**<<** "succes222s2"**<<endl**;

        x = p->data;

        ls = p->next;

**free**(p);

    }

        return 1;

}

int **main**(){

**LinkStack** \*ls;

**initstack**(ls);

    char s;

**push**(ls,'a');

**push**(ls,'e');

**push**(ls,'c');

**dispstack**(ls);

**pop**(ls,s);

**printf**("%c ",s);

**dispstack**(ls);

    return 0;

}

//队列

#include<stdio.h>

#include<cstdlib>

typedef char **ElemType**;

typedef struct **QNode**

{

**ElemType** data;

    struct **QNode** \*next;

}**QNode**;

typedef struct

{

**QNode** \*front;

**QNode** \*rear;

}**LinkQuene**;

void **initquene**(**LinkQuene** \*&lq){

    lq->front = lq->rear = **NULL**;

}

int **enquene**(**LinkQuene** \*&lq,**ElemType** x){

**QNode** \*s;

    s = (**QNode** \*)**malloc**(sizeof(**QNode**));

    s->data = x;

    s->next = **NULL**;

    if (lq->front == **NULL**){

        lq->front = lq->rear = s;

    }else{

        lq->rear->next = s;

        lq->rear = s;

    }

    return 1;

}

int **deQuene**(**LinkQuene** \*&lq,**ElemType** &x){

**QNode** \*s;

    s = (**QNode** \*)**malloc**(sizeof(**QNode**));

    if (lq->front == **NULL**){

        return 0;

    }

    s = lq->front;

    x = s->data ;

    if (lq->front == lq->rear){

        lq->front = lq->rear = **NULL**;

    }

    else{

        lq->front = lq->front->next;

    }

    return 1;

}

int **main**(){

**LinkQuene** \*lq;

**initquene**(lq);

}

1. 实验结果（简单演示说明与截图）

# 实验三 二叉树的操作

1.实验目的

1）掌握二叉树的链式存储结构的创建方法

2）掌握二叉树的先序、中序遍历方法

2.实验环境（软件环境）

Visual Studio Code

1. 实验内容（附上源代码及必要注释）

#include<stdio.h>

#include<iostream>

typedef int **ElemType**;

typedef struct **tnode**

{  **ElemType** data;

   struct **tnode** \*lchild,\*rchild; *//左右两个*

} **BTNode**;

*//默认规定一个传入的为最上面的头节点*

void **PreOrder**(**BTNode** \*b)

{  if (b!=**NULL**)

    {

**printf**("%c ",b->data);

**PreOrder**(b->lchild);

**PreOrder**(b->rchild);

    }

}

void **InOrder**(**BTNode** \*b)

{  if (b!=**NULL**)

    {   **InOrder**(b->lchild);

**printf**("%c ",b->data);

**InOrder**(b->rchild);

    }

}

void **PostOrder**(**BTNode** \*b)

{  if (b!=**NULL**)

    {   **PostOrder**(b->lchild);

**PostOrder**(b->rchild);

**printf**("%c ",b->data);

    }

}

int max=0;*//树高*

int **BT\_depth**(**BTNode** \*b,int depth)

{

    if(b)

    {

        if(b->lchild)

**BT\_depth**(b->lchild,depth+1);

        if(b->rchild)

**BT\_depth**(b->rchild,depth+1);

    }

    if(depth>max)

        max=depth;

    return depth;

}

int **main**(){

}

1. 实验结果（简单演示说明与截图）

# 实验四 图的操作

1.实验目的

1）掌握图的邻接矩阵和邻接表的创建方法

2）掌握图的深度遍历和广度遍历方法

2.实验环境（软件环境）

Visual Studio Code

1. 实验内容（附上源代码及必要注释）

//邻接矩阵

#include<stdio.h>

#include<iostream>

#define **MAXVEX** 100

typedef char **VertexType**;

int visited[**MAXVEX**];

*//两种方法 一种是顶点和边分开  一种是顶点和边合起来*

*//合起来*

typedef struct

{

**VertexType** Vex[**MAXVEX**];*//顶点表*

    int Edges[**MAXVEX**][**MAXVEX**]; *//边带权不带权均可*

    int vexnum, edgenum; *//顶点数和边数*

}**MGraph1**;

void **CreateGraph1**(**MGraph1** &g,int A[][**MAXVEX**],**VertexType** B[**MAXVEX**],int vexnum,int edgenum)

{

    g.vexnum = vexnum;

    g.edgenum = edgenum;

    for (int i = 0;i < vexnum; i++)

    {   for (int j = 0;j < vexnum; j++)

        {

            g.Edges[i][j] = A[i][j];

        }

    }

    for (int i = 0; i < **MAXVEX**; i++)

    {

        g.Vex[i] = B[i];

    }

}

*//顶点和边和起来*

typedef struct **vertex**

{

    int adjvex;

**VertexType** data;

}**VType**; *//顶点类型*

typedef struct **graph**

{

**VType** Vex[**MAXVEX**]; *//顶点集合*

    int Edges[**MAXVEX**][**MAXVEX**]; *//边的集合*

    int vexnum, edgenum; *//顶点数和边数*

}**MatGraph2**; *//图的邻接矩阵类型*

void **DFS**(**MGraph1** g, int i ){

    visited[i]=1;

**printf**("%c ", g.Vex[i]);

    for(int j = 0;j < g.vexnum;j++){

        if(g.Edges[i][j] == 1 && !visited[j]) *//边存在且顶点未被访问，递归*

**DFS**(g, j);

    }

}

void **DFSTraverse**(**MGraph1** g){

    for(int i = 0;i < g.vexnum ;i++)

        visited[i] = 0;

    for(int i = 0;i < g.vexnum;i++)

    {   if( !visited[i] )

**DFS**(g, i);

        }

}

int **main**(){

}

//邻接表

#include<stdio.h>

#include<iostream>

#define **MAXVEX** 100

#define **INF** 65536

typedef char **VertexType**;

typedef struct **edgenode**

{

    int adjvex; *//相邻点序号*

    int weight; *//边的权值*

    struct **edgenode** \*nextarc; *//下一条边的顶点*

} **ArcNode**; *//边结点*

typedef struct **vexnode**

{  **VertexType** data;

**ArcNode** \*firstarc;

} **VHeadNode**; *//表头指针*

typedef struct

{   int vexnum, edgenum; *//顶点数和边数*

**VHeadNode** adjlist[**MAXVEX**]; *//单链表头结点数组*

}**AdjGraph**; *//邻接表*

*//将邻接矩阵转化成邻接表*

void **CreateGraph**(**AdjGraph** \*&G,int A[][**MAXVEX**],int vexnum,int edgenum)

{

**ArcNode** \*p;

    G=(**AdjGraph** \*)**malloc**(sizeof(**AdjGraph**));

    G->vexnum = vexnum;

    G->edgenum = edgenum;

    for (int i = 0;i < G->vexnum;i++)

        G->adjlist[i].firstarc=**NULL**;

    for (int i = 0;i < G->vexnum;i++)

        for (int j= G->vexnum - 1;j >= 0;j--)

        {

            if (A[i][j]>0 && A[i][j]<**INF**)

            {

                p=(**ArcNode** \*)**malloc**(sizeof(**ArcNode**)); *//创建结点p*

                p->adjvex = j;

                p->weight = A[i][j];

                p->nextarc = **NULL**;

*//结点始终在在最后一个*

                p->nextarc = G->adjlist[i].firstarc; *//头插法插入p*

                G->adjlist[i].firstarc = p;

            }

        }

}

*//广度遍历   vi是初始节点*

void **BFS**(**AdjGraph** \*G,int vi)

{

    int v,visited[**MAXVEX**];

**ArcNode** \*p;

    int Qu[**MAXVEX**];*//循环队列*

    int front=0,rear=0;

    for (int i = 0;i < G->vexnum ;i++)

    {    visited[i]=0;}

**printf**("%d ",vi); *//访问初始顶点*

    visited[vi]=1;

    rear = (rear+1) % **MAXVEX**;

    Qu[rear] = vi; *//初始顶点进队*

    while (front != rear)

    {

        front = (front+1) % **MAXVEX**;

        v = Qu[front]; *//出队顶点v*

        p = G->adjlist[v].firstarc; *//v的第一个相邻点*

        while (p != **NULL**) *//v的所有相邻点*

        {

            if ( visited[p->adjvex] == 0)

            {

**printf**("%d ",p->adjvex);

                visited[p->adjvex] = 1;

                rear = (rear+1) % **MAXVEX**;

                Qu[rear] = p->adjvex;

            }

            p = p->nextarc; *//一层一层来*

        }

    }

}

int **main**(){

}

4.实验结果（简单演示说明与截图）

# 

# 实验五 查找操作

1.实验目的

1) 掌握二分查找算法

2) 掌握二叉排序树的生成、插入、查找等算法

2.实验环境（软件环境）

Visual Studio Code

3.实验内容（附上源代码及必要注释）

//二分查找

#include<stdio.h>

#include<iostream>

using namespace **std**;

typedef int **KeyType**;

typedef char **ElemType**;

typedef struct

{   **KeyType** key; *// 关键字*

**ElemType** data;

}**SqType**;

int **BinSearch**(**SqType** R[],int n,**KeyType** k) *//n是数组大小*

{

    int low=0,high=n-1,mid;

    while (low<=high) *//循环*

    {

        mid=(low+high)/2;

        if (R[mid].key==k)

            return mid+1; *//返回逻辑序号*

        else if (R[mid].key>k) *//在R[low..mid-1]中查找*

            high=mid-1;

        else

            low=mid+1; *//在R[mid+1..high]中查找*

    }

   return 0; *//没有找到返回0*

}

**status** **DeleteBST**(**BiTree**\* T, int key)

{

    if (!\*T)

        return false;

    else

    {

        if (key == (\*T)->data)

        {

            return **Delete**(T);

        }

        else if (key<(\*T)->data)

        {

            return **DeleteBST**(&(\*T)->lchild,key);

        }

        else

        {

            return **DeleteBST**(&(\*T)->rchild,key);

        }

    }

}

//二叉排序树

#include<cstdio>

#include<iostream>

using namespace **std**;

typedef int KeyType;

typedef char ElemType;

typedef struct **BSTNode**

{   KeyType key;

    ElemType data;

    struct **BSTNode** \*lchild,\*rchild;

}**BSTNode**,\***BSTree**;

**BSTree** **SearchBST**(**BSTree** T,**KeyType** key) *//递归查找*

{

    if((!T)||key == T->key){

        return T;

    }else if(key < T->key){

        return **SearchBST**(T->lchild,key);

    }else{

        return **SearchBST**(T->rchild,key);

    }

}

void **InsertBST**(**BSTree** &T,**KeyType** key)

{ *//当二叉排序树T中不存在关键字等于e.key的数据元素时，则插入该元素*

    if(!T)

    {

        BSTree S;

        S->key = key;

        S->lchild = S->rchild = NULL;

        T = S; *//T为空时才赋值*

    }else if( key < T->key){

**InsertBSt**(T->lchild,key);

    }else if( key > T->key){

**InsertBST**(T->rchild,key);

    }

}

void **CreatBST**(**BSTree** &T,**KeyType** a[],int n)

{

    T = NULL;

    for(int i = 0;i < n;i++){

**InsertBST**(T,a[i]);

    }

}

int **DeleteBST**(**BiTree** &T, int key)

{

    if (!T)

        return 0;

    else

    {

        if (key == T->key)

        {

            return **Delete**(T);

        }

        else if ( key< T->key)

        {

            return **DeleteBST**(T->lchild,key);

        }

        else

        {

            return **DeleteBST**(T->rchild,key);

        }

    }

}

*// 目标结点是一个叶子结点，直接删除即可*

*// 目标结点存在左子树，用其左子树里的最大结点覆盖掉目标结点*

*// 目标结点存在右子树，用其右子树里的最小结点覆盖掉目标结点*

int **Delete**(**BiTree** &p)

{

    BiTree q, s;

    if (p->rchild==NULL)

    {

        q = p;

        p = p->lchild;

**free**(q);

    }

    else if (p->lchild==NULL)

    {

        q = p;

        p = p->rchild;

**free**(q);

    }

    else

    { *//左右都不为空*

        q = p;

        s = p->lchild;

        while (s->rchild)

        { *//左子树的最右值 (最大值)*

            q = s;

            s = s->rchild;

        }

        p->key = s->key;

*//转移了放右边  为转移放左边*

        if (q != p)

            q->rchild = s->lchild;

        else

            q->lchild = s->lchild;

**free**(s);

    }

    return 1;

}

int **main**(){

}

4.实验结果（简单演示说明与截图）

# 

# 实验六 排序操作

1.实验目的

1）掌握直接插入排序算法

2）掌握简单选择排序算法。

2.实验环境（软件环境）

Visual Studio Code

1. 实验内容（附上源代码及必要注释）

#include<cstdio>

#include<iostream>

using namespace **std**;

typedef int KeyType;

typedef char ElemType;

typedef struct

{   KeyType key;

    ElemType data;

}**SqType**;

void **InsertSort**(**SqType** R[],int n)

{   int j;

    SqType tmp;

    for (int i = 1;i < n;i++) *//从第二个元素即R[1]开始的*

    {

        if (R[i-1].key > R[i].key)

        {

            tmp = R[i];

            j = i-1; *//在R[0..i-1]中找R[i]的插入位置*

            do

            {

                R[j+1]=R[j]; *//将关键字大于tmp.key的元素后移*

                j--; *//继续向前比较*

            }while(j>=0 && R[j].key>tmp.key);

            R[j+1]=tmp; *//在j+1处插入R[i]*

        }

    }

}

void **SelectSort**(**SqType** R[],int n)

{   int k;

    SqType tmp;

    for (int i = 0;i < n-1; i++)

    {

        k = i;

        for (int j = i+1;j < n;j++)

        {   if (R[j].key<R[k].key)

                k = j; *//用k指出每趟在无序区段的最小元素*

        }

        if (k!=i)

        {

            tmp=R[i]; *//将R[k]与R[i]交换*

            R[i]=R[k];

            R[k]=tmp;

        }

    }

}

void **QuickSort**(**SqType** R[],int s,int t)

{   int i=s,j=t;

    SqType tmp;

    if (s<t)

    {

        tmp=R[s];

        while (i!=j) *//从区间两端交替向中间扫描,直至i=j*

        {   while (j>i && R[j].key>=tmp.key)

            j--; *//从右向左,找第1个小于tmp.key的R[j]*

            R[i]=R[j]; *//将R[j]前移到R[i]的位置*

            while (i<j && R[i].key<=tmp.key)

                i++; *//从左向右扫描,找第1个大于tmp.key的R[i]*

                R[j]=R[i]; *//将R[i]后移到R[j]的位置*

        }

        R[i]=tmp;

**QuickSort**(R,s,i-1);

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

    }

}

4.实验结果（简单演示说明与截图）