数据结构实验报告

原创

学习吧TOTORO 于 2019-01-12 16:22:02 发布 20918 收藏 208
分类专栏： 数据结构 文章标签： 数据结构实验报告

转载请注明出处，代码可以直接用，实验截图未上传成功，在编译器里运行一下就有！！！！

南通大学计算机科学与技术学院

数据结构

实验报告书

实 验 名__数据结构实验报告__

班　级___网络工程***_____

姓　名_____**_____

学　号_____*******_____

指导教师____******_____

日　期___2018/01/16____

目 录

实验一　约瑟夫环问题

1 实验目的：

为了更好的巩固有关链表的知识点，在实际操作中掌握循环单链表的相关应用，夯实基础，掌握学习内容中的重要点，将理论与实践更好的结合，将用循环单链表来解决经典问题——约瑟夫环问题。

2 算法思想：

约瑟夫环问题的存储结构，由于约瑟夫环问题本身具有循环性，考虑采用循环链表，为了统一对表中任意结点的操作，循环链表不带头结点，将循环链表的结点定义为如下结构：

```
Struct {
    Int data;
    Node *next;
   };
```

3 程序设计：

下面给出伪代码描述：

①工作指针pre和p初始化，计数器初始化；

Pre=head;p=head->next;

②循环直到p=pre ;

如果count=m,则输出结点p的编号；

删除结点p,即p=pre->next;

否则执行工作指针pre 和p后移；

③退出循环，链表中仅剩下一个结点p,输出结点后删除结点；

4程序试调及结果 ：

附源程序：

```
#include<iostream>
using namespace std;
struct Node
{
    int data;
    int number;
    Node *next;
```

```
};
int main()
{
    struct Node *Head, *pre, *p;
    int n,m;
    cout<<"请输入总人数n:";
    cin>>n;
    cout<<"请输入要删除编号m:";
    cin>>m;
    Head=pre=new Node;
    Head->number=1;
    cout<<"请输入第一个数:";
    cin>>Head->data;
    cout<<"请输入最后一个数:";
    for(int i=2;i<n;i++)
     pre->next=new Node;
     pre=pre->next;
     cin>>pre->data;
     pre->number=i;
    pre->next=Head;
    cout<<"将退出循环的数是:"<<endl;
    while(n)
    {
        for(int j=1;j<m;j++)
        pre=pre->next;
        p=pre->next;
        pre->next=p->next;
        cout<<p->data;
        delete p;
        n--;
```

```
}
return 0;}
```

5总结：

对链表的操作仍需要进一步深入了解，在上机过程中，对于要退出循环的编号所对应的结点的操作仍存在问题，考虑到时循环链表，两个指针的分工也及其重要，要多加练习！

实验二　　学生信息管理

1实验目的：

为进一步巩固所学知识并将其应用在实际生活中，已达到学以致用的目的，我这次做的学生信息管理的实验是用大一时VC++中的文件输入输出流来做的，本来打算将文件输入输出流与数据结构中的链表相结合操作，能力有限，仅用大一所学知识在此实验报告中！

2问题描述：

将学生的学号 姓名 成绩分别输入，并根据成绩分数进行输出依次为学号 姓名 成绩。（排序部分的功能未写出）

3知识回顾：

①文件流的使用方法：

Ofstream outfile;

Fstream iofile;

②使用文件流对象的成员函数close()/open()关闭文件/打开文件

4 程序试调及结果：

```
#include<fstream.h>
#include<iostream.h>
#include<iomanip.h>
class Std
{
    int no;
    char name[10];
    int degree;
public:
```

```
        void gdata()
        {
                cout<<"(学号 姓名 成绩):";
            cin>>no>>name>>degree;
        }
        void display()
        {
                cout<<setw(6)<<no<<setw(6)<<name<<setw(6)<<degree<<endl;
        }
};
void func1()
{
        ofstream out;
        out.open("std.dat");
        Std s;
        int n;
        cout<<"输入数据"<<endl;
        cout<<"学生人数:";
        cin>>n;
        for(int i=0;i<n;i++)
        {
                cout<<" 第"<<i+1<<"个学生";
                s.gdata();
                out.write((char*)&s,sizeof(s));
        };
        out.close();
}
void func2()
{
        ifstream in;
        in.open("std.dat");
```

```
    Std s;
    cout<<"输入数据"<<endl;
    cout<<"学号 姓名 成绩"<<endl;
    in.read((char*)&s,sizeof(s));
    while(in)
    {
        s.display();
        in.read((char*)&s,sizeof(s));
    }
    in.close();
}
void main()
{
    int n;
    do
    {
        cout<<"选择(1:输入数据 2:输出数据 其他:退出):";
        cin>>n;
        switch(n)
        {
        case 1:func1();break;
        case 2:func2();break;
        }
    }while(n==1||n==2);
}
```

5 实验总结：

写本次实验报告与数据结构中涉及的算法并无太多关联，但知识之间是相互融会贯通VC++是数据结构算法能够熟练运用的基石，它是实践代码的重要工具语言，因此不可学新忘旧，所学知识一定要经常反复练习，才会获得更多收获和成长！

实验三　　二叉树的建立与遍历

1 实验目的：

编写程序，实现二叉树的建立，并实现先序、中序和后序遍历。如：输入先序序列abc###de###，则建立二叉树。显示其先序序列为：abcde，中序序列为:cbaed, 后序序列为：cbeda。将二叉树的实际应用融汇在实践学习中，切勿纸上谈兵，此题仅仅是对二叉树的建立和遍历，没有进行具体的应用，意在掌握二叉树的的基本思想及遍历过程。

2 算法思想：

二叉树的遍历思想，即沿着某条搜索路线，依次对树中每个结点均做一次访问，由二叉树的定义可知，一棵非空二叉树由根结点及左右子树这三个部分组成，因此，在任一给定的结点上可以按照某种次序执行以下操作：

1. 访问根节点D；
2. 访问左子树L；
3. 访问右子树R；

先序序列：DLR 中序序列：LDR 后序序列：LRD

3 程序设计：

```
char data;
    Btnode *lchild;
    Btnode *rchild;
//定义左右孩子
   Btnode *bt;
public:
   BinaryTree() { bt = NULL;}
   ~BinaryTree() { clear(bt); }
   int clear(Btnode *bt);
   void CreateBinaryTree(char end);
   int create(Btnode *p_work, int k, char end);
   void PreTraBiTree();
   int PreTraverse(Btnode *p_work);
   void lnTraBiTree();
   int lnTraverse(Btnode *p_work);
   void PostTraBiTree();
   int PostTraverse(Btnode *p_work);
create(p_new, 0, end);
```

```
    create(p_new, 1, end);
```

//定义指针和二叉树，再分别用递归算法定义先序中序后序遍历算法，创建根结点左子树和右子树，0是标志为左子树，1为右子树

```
int BinaryTree::PreTraverse(Btnode *p_work)

int BinaryTree::InTraverse(Btnode *p_work)

int BinaryTree::PostTraverse(Btnode *p_work)
```

//再分别创建先序中序后序遍历运算

4 程序及其调试结果

```
#include<iostream>

using namespace std;


struct Btnode

{

    char data;

    Btnode *lchild;

    Btnode *rchild;

};


class BinaryTree {

    Btnode *bt;

public:

    BinaryTree() { bt = NULL;   }

    ~BinaryTree() { clear(bt); }

    int clear(Btnode *bt);

    void CreateBinaryTree(char end);

    int create(Btnode *p_work, int k, char end);

    void PreTraBiTree();

    int PreTraverse(Btnode *p_work);

    void InTraBiTree();

    int InTraverse(Btnode *p_work);
```

```
    void PostTraBiTree();
    int PostTraverse(Btnode *p_work);
};

int BinaryTree::clear(Btnode *bt)
{
    if (bt)
    {
        clear(bt->lchild);
        clear(bt->rchild);
        delete bt;
    }
    return 0;
    bt = NULL;
}void BinaryTree::CreateBinaryTree(char end)
{
    cout << "输入先序序列，\"#\"为空指针域标志：";
    Btnode *p_new;
    char x;
    cin >> x;
    if (x == end)return;
    p_new = new Btnode;
    if (!p_new)
    {
        cout << "申请内存失败！" << endl;
        exit(-1);
    }
    p_new->data = x;
    p_new->lchild = NULL;
    p_new->rchild = NULL;
    bt = p_new;
```

```
    create(p_new, 0, end);
    create(p_new, 1, end);
}

int BinaryTree::create(Btnode *p_work, int k, char end)
{
    Btnode *p_new;
    char x;
    cin >> x;
    if (x != end)
    {
        p_new = new Btnode;
        p_new->data = x;
        p_new->lchild = NULL;
        p_new->rchild = NULL;
        if (k == 0)p_work->lchild = p_new;
        if (k == 1)p_work->rchild = p_new;
        create(p_new, 0, end);
        create(p_new, 1, end);
    }
    return 0;
}

void BinaryTree::PreTraBiTree()
{
    cout << "先序遍历：";
    Btnode *p;
    p = bt;
    PreTraverse(p);
    cout << endl;
}
```

```
int BinaryTree::PreTraverse(Btnode *p_work)
{
    if (p_work)
    {
        cout << p_work->data << " " ;
        PreTraverse(p_work->lchild);
        PreTraverse(p_work->rchild);
    }
    return 0;
}

void BinaryTree::InTraBiTree()
{
    cout << "中序遍历：";
    Btnode *p;
    p = bt;
    InTraverse(p);
    cout << endl;
}
int BinaryTree::InTraverse(Btnode *p_work)
{
    if (p_work)
    {
        InTraverse(p_work->lchild);
        cout << p_work->data << " " ;
        InTraverse(p_work->rchild);
    }
    return 0;
}
```

```
void BinaryTree::PostTraBiTree()
{
    cout << "后序遍历：";
    Btnode *p;
    p = bt;
    PostTraverse(p);
    cout << endl;
}

int BinaryTree::PostTraverse(Btnode *p_work)
{
    if (p_work)
    {
        PostTraverse(p_work->lchild);
        PostTraverse(p_work->rchild);
        cout << p_work->data << " " ;
    }
    return 0;
}

int main()
{
    BinaryTree Tree;
    Tree.CreateBinaryTree('#');
    Tree.PreTraBiTree();
    Tree.InTraBiTree();
    Tree.PostTraBiTree();
    return 0;

}
```

5 总结：

实验感想：二叉树的性质容易懂却很难记忆，对二叉树的存储结构和遍历算法这部分内容掌握要更加努力专研，才能够熟练运用。我会严格要求自己，熟练掌握算法思想，尽量独立完成程序的编写与修改工作，只有这样，才能够提高运用知识，解决问题的能力。

实验四　　最短路径问题

1 实验目的：

最短路劲问题是图结构型结构的重点应用，在现实生活中的应用十分广泛，城市交通导航问题都要运用到最短路劲的算法思想去实现，它起着不可或缺的重要作用，在解决最短路径的问题过程中可以采用DISJKSTRA算法和FLOYD算法，将其在实践中加以应用用。

2 算法思想：

最短路径问题就是指在带权值的地图中， 寻找从指定起点到终点的一条具有最小权值总和的路径问题。如果把权值看成是道路的长度属性， 那么目标路径就是从起点到终点的最短路径。所谓最短路径就是网络中两点之间距离最短的路径，这里讲的距离可以是实际的距离，最短路径不仅仅指一般地理意义上的距离最短，也可以引申为其它的度量，如时间、运费、流量等。在求解最短路劲问题中将使用dijkstra算法来解决问题，dijkstra算法的基本思想是：按距离由近到远为顺序，依次求得g的各顶点的的最短路和距离，为避免重复并保留每一步的计算信息,直至算法结束。

3 程序设计：

```
struct ArcCell{
    int adj;//该弧所指向的顶点的位置


    char *info;//该弧相关信息的指针
};


struct VertexType{
    int number;
    char *view;
};
```

```
struct MGraph{
    VertexType vex[NUM];
    ArcCell arcs[NUM][NUM];
    int vexnum,arcnum;
```

4 程序及其调试结果：

```
#include<iostream>
#include<stdio.h>
using namespace std;
#define Max 5000
#define NUM 17
struct ArcCell{
    int adj;
    char *info;
};
struct VertexType{
    int number;
    char *view;
};
struct MGraph{
    VertexType vex[NUM];
    ArcCell arcs[NUM][NUM];
    int vexnum,arcnum;
};
MGraph g;
int P[NUM][NUM];
long int D[NUM];
int x[13]={0};
void CreateUDN(int v,int a);
void ShortestPath(int num);
```

```
void output(int view1,int view2);
char Menu();
void CreateUDN(int v,int a)
{
    int i,j;
    g.vexnum=v;
    g.arcnum=a;
    for(i=1;i<g.vexnum;++i)
    g.vex[i].number=i;
    for(i=1;i<g.vexnum;++i)
    g.vex[i].number=i;
    g.vex[0].view="名称";
    g.vex[1].view="西门";
    g.vex[2].view="西操场";
    g.vex[3].view="一食堂";
    g.vex[4].view="北门";
    g.vex[5].view="一超市";
    g.vex[6].view="图书馆";
    g.vex[7].view="校园服务中心";
    g.vex[8].view="小北街";
    g.vex[9].view="纺化楼";
    g.vex[10].view="二超市";
    g.vex[11].view="逸夫楼";
    g.vex[12].view="二食堂";
    g.vex[13].view="范曾艺术馆";
    g.vex[14].view="体育馆";
    g.vex[15].view="东操场";
    g.vex[16].view="医疗服务中心";
    for(i=1;i<g.vexnum;++i)
    {
        for(j=1;j<g.vexnum;++j)
```

```
        {
            g.arcs[i][j].adj=Max;
        }
    }
    g.arcs[1][2].adj=g.arcs[2][1].adj=200;
    g.arcs[1][3].adj=g.arcs[3][1].adj=400;
    g.arcs[1][9].adj=g.arcs[9][1].adj=600;
    g.arcs[2][4].adj=g.arcs[4][2].adj=300;
    g.arcs[3][5].adj=g.arcs[5][3].adj=500;
    g.arcs[3][6].adj=g.arcs[6][3].adj=200;
    g.arcs[4][8].adj=g.arcs[8][4].adj=30;
    g.arcs[4][5].adj=g.arcs[5][4].adj=100;
    g.arcs[5][6].adj=g.arcs[6][5].adj=50;
    g.arcs[5][7].adj=g.arcs[7][5].adj=200;
    g.arcs[6][7].adj=g.arcs[7][6].adj=100;
    g.arcs[7][10].adj=g.arcs[10][7].adj=30;
    g.arcs[9][3].adj=g.arcs[3][9].adj=500;
    g.arcs[10][12].adj=g.arcs[12][10].adj=10;
    g.arcs[10][11].adj=g.arcs[11][10].adj=300;
    g.arcs[12][16].adj=g.arcs[16][12].adj=20;
    g.arcs[12][13].adj=g.arcs[13][12].adj=700;
    g.arcs[12][14].adj=g.arcs[14][12].adj=600;
    g.arcs[13][15].adj=g.arcs[15][13].adj=400;
    g.arcs[14][15].adj=g.arcs[15][14].adj=100;

}
void ShortestPath(int num)
{                       //最短路径
    int v,w,i,t;
    int final[NUM];
    int min;
```

```
for(v=1;v<NUM;v++)
{
    final[v]=0;
    D[v]=g.arcs[num][v].adj;
    for(w=1;w<NUM;w++)
        P[v][w]=0;
    if(D[v]<32767)
    {
        P[v][num]=1;
        P[v][v]=1;
    }
}
D[num]=0;
final[num]=1;
for(i=1;i<NUM;++i)
{
    min=Max;
    for(w=1;w<NUM;++w)
        if(!final[w])
            if(D[w]<min)
            {
                v=w;
                min=D[w];
            }
            final[v]=1;
            for(w=1;w<NUM;++w)
                if(!final[w]&&((min+g.arcs[v][w].adj)<D[w]))
                {
                    D[w]=min+g.arcs[v][w].adj;
                    for(t=0;t<NUM;t++)
                        P[w][t]=P[v][t];
```

```
                P[w][w]=1;
            }
    }
}

void output(int view1,int view2)
{                       //全部最短路径
    int a,b,c,d,q=0;
    a=view2;
    if(a!=view1)
    {
        cout<<"从"<<g.vex[view1].view<<"到"<<g.vex[view2].view<<'\n'
            <<"最短距离为" <<D[a]<<'\t' <<g.vex[view1].view;
        d=view1;
        for(c=0;c<NUM;++c)
        {
            gate:;
            P[a][view1]=0;
            for(b=0;b<NUM;b++)
            {
                if(g.arcs[d][b].adj<32767&&P[a][b])
                {
                    cout<<"-->"<<g.vex[b].view;
                    q=q+1;
                    P[a][b]=0;
                    d=b;
                    if(q%8==0) cout<<'\n';
                    goto gate;
                }
            }
        }
```

```
        cout<<'\n';
    }
}
void place()
{
    cout<<"  景点枚举：\n"
        <<"  1:西门     2:西操场     3:一食堂     4:北门        5:一超市\n"
        <<"  6:图书馆    7:校园服务中心        8:小北街        9:纺化楼     10:二超市\n"
        <<"  11:逸夫楼    12:二食堂       13:范曾艺术馆    14:体育馆    15:东操场  16:医疗服务中心"<<endl;
}
int main()
{
    int v0,v1;
    char ck1;
    CreateUDN(NUM,17);
    do
    {
            place();
            cout<<"请选择出发地序号（1～16）：";
            cin>>v0;
            cout<<"请选择目的地序号（1～16）：";
            cin>>v1;
            while(v1>16||v1<1){
               cout<<"输入的目的地序号错误v1 error"<<'\n';
               cout<<"请重新选择目的地序号（1～16）：";
               cin>>v1;
            }
            ShortestPath(v0);
            output(v0,v1);
            cout<<"请按回车键返回主菜单"<<'\n';
            getchar();
```

```
            getchar();
            break;

    }while(ck1!='1');
return 0;

}
```

5 总结：

在编程过程中一定要多思考，遇到困难和错误要有耐心一个一个慢慢解决，任何事情想要做好都不是轻而易举就能完成的，必须要付出相应的时间，精力去完成，可能有的时候写了十行代码就有了一百多个错误，都不知道那些错误要从哪里下手解决，很打击自己的自信心，但是有的时候我们只要从头到尾的把思路思绪理清楚，不懂得知识点问老师同学上网查资料，相信总会有办法解决，要注重逻辑思维的养成，学会站在计算机的角度去思考问题，从而解决问题，多多上机实践，多犯错误，然后再不断地改正错误，我相信自己会收获更多，也会成长的更快，在解决最短路劲问题的算法中，对于有向完全图的操作，有向网无向网的操作也是掌握的还不够熟练，应加强训练，从实践中获得成长！

实验五　　HASH查找问题

1 实验目的：

为进一步巩固查找这一单元的知识，更加深刻理解查找中主要运用到的思想和操作过程，并熟悉查找中所用到不同的方法如静态/动态查找有序表的折半查找，分块查找，动态查找的操作过程和算法思想，达到学以致用的目的。

2 算法思想：

选取某个函数，依该函数按关键字计算元素的存储位置，并按此存放，

由同一个函数对给定值K计算地址，将K与地址单元中元素关键字进行比较，确定查找是否成功，这个就是哈希方法，按哈希思想构造出来的表叫哈希表，哈希查找的本质是先将数据映射成它的哈希值；哈希查找的核心是构造一个查找时，哈希函数，它将原来直观、整洁的数据映射为看上去似乎是随机的一些整数。

哈希查找的操作步骤：

1)　用给定的哈希函数构造哈希表；

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。

3 程序设计：

建立哈希表操作步骤：

1) step1 取数据元素的关键字key，计算其哈希函数值（地址）。若该地址对应的存储空间还没有被占用，则将该元素存入；否则执行step2解决冲突。

2) step2 根据选择的冲突处理方法，计算关键字key的下一个存储地址。若下一个存储地址仍被占用，则继续执行step2，直到找到能用的存储地址为止。

哈希查找步骤为：

1) Step1 对给定k值，计算哈希地址 Di=H（k）；若HST为空，则查找失败；若HST=k，则查找成功；否则，执行step2（处理冲突）。

2) Step2 重复计算处理冲突的下一个存储地址 Dk=R（Dk-1），直到HST[Dk]为空，或HST[Dk]=k为止。若HST[Dk]=K，则查找成功，否则查找失败。

4 程序及其调试结果：

```
#include<iostream>
#include<cmath>
using namespace std;
#define SUCCESS 1;
#define UNSUCCESS 0;
```

```
#define NULLKEY -1;
#define TableLength 13;
#define p 13;// H(key)=key % p
typedef int T;
template <class T>
struct ElemType
{
    T key;//关键字
};
template <class T>
class LHSearch
{
private:
    ElemType<T> *HT; //开放定址哈希表
    int count; //当前数据元素个数
    int size; //哈希表长度
public:
     LHSearch(); //
    ~LHSearch(); //
    void lnitHashTable(int n);//
    int Hash(T key); //计算哈希地址
    void Collision(int &s);//冲突,计算下一个地址
    int Search(T key,int &s);//哈希查找
    int lnsert(ElemType<T> e); //元素插入
    void Display(); //显示哈希表
};
template <class T>
LHSearch<T>::LHSearch()
{
    HT=NULL;
    size=0;
```

```
    count=0;
}
template <class T>
LHSearch<T>::~LHSearch()
{ delete [] HT;
  count=0;
}
template <class T>
int LHSearch<T>::Hash(T key)
{//由哈希函数求哈希地址
    return key%p;
}
template <class T>
void LHSearch<T>::Collision(int &s)
{//开放定址法解决冲突
    s=s++;
}
template <class T>
int LHSearch<T>::Search(T key,int &s)
{//查找，找到返回
    //int s;
    s=Hash(key);
    while((HT[s].key!=-1) && (key!=HT[s].key))
        Collision(s);
    if(HT[s].key==key)
        return 1;
    else
        return 0;
}
template <class T>
int LHSearch<T>::Insert(ElemType<T> e)
```

```
{//插入元素
    int s;
    if(count==size)
    {
        cout<<"表满，不能插入!"<<endl;
        return UNSUCCESS;
    }
    else
    {
        s=Hash(e.key);
        int f;
        f=Search(e.key,s);
        if(f)   //表中已有和e的关键字相同的元素,不进行插入操作
        {
           cout<<"该元素已存在，不能插入!"<<endl;
         return UNSUCCESS; }
       else
       {
          HT[s].key=e.key;
          cout<<"插入成功!"<<endl;
          count++;
         return SUCCESS; }
     }
}
template<class T>
void LHSearch<T>::lnitHashTable(int n)
{
    size=n;
    HT=new ElemType<T>[size];
    for(int i=0;i<size;i++) //初始化，把哈希表置空
        HT[i].key=NULLKEY;
```

```
}
template<class T>
void LHSearch<T>::Display()
{
    for(int i=0;i<size;i++)
    {
        cout<<i<<'\t';
        if(HT[i].key!=-1)
            cout<<HT[i].key;
        else
            cout<<'\t';
        cout<<endl;
    }
}
void main()
{
    int m;
    T key;
    int s=0;
    ElemType<int> e;
    LHSearch<int> a;
    cout<<"输入相应代码，必须先创建哈希表)\n";
    do {
        cout<<"--- 1. 创建查找表  --\n"
            <<"--- 2. 插入----------\n"
            <<"--- 3. 查找----------\n"
            <<"--- 4. 显示 ---------\n"
            <<"--- 5. 退出 ---------\n"
            <<"请选择操作:";
            cin>>m;
            switch(m)
```

```
        {
        case 1://创建查找表
            cout<<"请输入表容量：\n";
            cin>>m;
            a.InitHashTable(m);
            cout<<"依次输入表元素，-1结束:\n";
            for(cin>>e.key;e.key!=-1;cin>>e.key)
                a.Insert(e);
            break;
        case 2:  //插入元素
            cout<<"输入元素:\n";
            cin>>e.key;
            a.Insert(e);
            break;
        case 3: //查找
            cout<<"请输入查找关键字：\n";
            cin>>key;
            if(a.Search(key,s))
                cout<<"找到!\n";
            else
                cout<<"不存在，末找到!\n";
            break;
        case 4://显示哈希表
            a.Display();
            break;
        case 5://
           cout<<"结束!\n";
           break;
        }
    }while(m!=5);
}
```

5 实验总结：

顺序查找的原理很简单，就是遍历整个列表，逐个进行记录的关键字与给定值比较，若某个记录的关键字和给定值相等，则查找成功，找到所查的记录。如果直到最后一个记录，其关键字和给定值比较都不等时，则表中没有所查的记录，查找失败。

折半查找技术，也就是二分查找。它的前提是线性表中的记录必须是关键码有序（通常从大到小有序），线性表必须采用顺序存储。折半查找的基本思想是：取中间记录作为比较对象，若给定值与中间记录的关键字，则在中间记录的关键字相等，则查找成功；若给定值小于中间记录的作伴去继续查找；若给定值大于中间记录的关键字，则在中间记录的右半区继续查找。不断重复上述过程，直到查找成功，或所有查找区域无记录，查找失败为止。

哈希表不可避免冲突现象：对不同的关键字可能得到同一哈希地址 即key1≠key2，而hash(key1)=hash(key2）。具有相同函数值的关键字对该哈希函数来说称为同义词。因此，在建造哈希表时不仅要设定一个好的哈希函数，而且要设定一种处理冲突的方法。

实验学习中，我发现了很多自己对于程序理解的错误，所以在调试时，出现了很多的错误，在一次次的改正错误的过程中，我渐渐加深了自己对于算法的理解，但是，还存在一定的问题，在运用上还有很大的不足，希望在以后的练习中，渐渐掌握。

实验六　　排序问题

1 实验目的：

1. 掌握排序的有关概念和特点。
2. 熟练掌握直接插入排序，希尔排序，冒泡排序，快速排序，简单选择排序，堆排序，归并排序，技术排序等的基本思想。
3. 关键字序列有序与无序，对于不同的排序方法有不同的影响，通过该实验进一步加深理解。

2 问题描述：

编写顺序查找程序，对一下数据查找37所在的位置：

5 13 19 21 37 56 64 75 80 88 92

编写折半查找程序，对一下数据查找37所在的位置：

5 13 19 21 37 56 64 75 80 88 92

分别用三种排序分别实现。

3 算法设计：

编译环境：visual studio 2017

```
struct SqList {//结构体实现
int * key;
int length;
};
```

主要算法设计：

(1)希尔排序

```
void ShellInsert(SqList &L, int dk)
{//对顺序表进行一趟希尔插入排序
for (int i = dk + 1;i <= L.length;i++)
{
	int j = 0;
	if (L.key[i] <= L.key[i - dk])
	{
		L.key[0] = L.key[i];
		for (j = i - dk;j > 0 && L.key[0] <= L.key[j];j -= dk)
			L.key[j + dk] = L.key[j];
		L.key[j + dk] = L.key[0];
	}
}
}

void ShellSort(SqList &L, int dlta[], int t)
{
//按增量序列dl[0]--dl[t-1]对顺序表L作哈希排序
for (int k = 0;k<t;k++)
	ShellInsert(L, dlta[k]);
}
```

(2)快速排序

```
//（1）起泡排序


void BubbleSort(SqList&L)
{
for (int i = 1;i<L.length;i++)//用i控制比较趟数共n-1趟
{
	int t;
	for (int j = 1;j <= L.length - i;j++)
		if (L.key[j]>L.key[j + 1])
		{
			t = L.key[j];
			L.key[j] = L.key[j + 1];
			L.key[j + 1] = t;
		}
}
}


int SelectMinKey(SqList& L, int n)
{
int min = n;
int minkey;//最小值
minkey = L.key[n];
for (int i = n + 1;i <= L.length;i++)
	if (L.key[i]<minkey)
	{
		minkey = L.key[i];
		min = i;
	}
return min;
}
```

(3)简单选择排序

```
void SelectSort(SqList& L)
{
//对顺序表L作简单选择排序
int j;
int t;
for (int i = 1;i <= L.length;i++)
{
	j = SelectMinKey(L, i);//在L.key[i]--L.key[L.length]中选择最小的记录并将其地址赋给j
	if (i != j)//交换记录
	{
		t = L.key[i];
		L.key[i] = L.key[j];
		L.key[j] = t;
	}
}
}
```

4 程序调试及其结果：

```
#include "stdafx.h"
//第十章 内部排序
#include<iostream>
using namespace std;


struct SqList {
int * key;
int length;
};
```

```
//(4)希尔排序

void ShellInsert(SqList &L, int dk)
{//对顺序表进行一趟希尔插入排序
for (int i = dk + 1;i <= L.length;i++)
{
    int j = 0;
    if (L.key[i] <= L.key[i - dk])
    {
        L.key[0] = L.key[i];
        for (j = i - dk;j > 0 && L.key[0] <= L.key[j];j -= dk)
            L.key[j + dk] = L.key[j];
        L.key[j + dk] = L.key[0];
    }
}
}

void ShellSort(SqList &L, int dlta[], int t)
{
//按增量序列dl[0]--dl[t-1]对顺序表L作哈希排序
for (int k = 0;k<t;k++)
    ShellInsert(L, dlta[k]);
}
//二快速排序
//（1）起泡排序

void BubbleSort(SqList&L)
{
for (int i = 1;i<L.length;i++)//用i控制比较趟数共n-1趟
{
```

```
    int t;
    for (int j = 1;j <= L.length - i;j++)
        if (L.key[j]>L.key[j + 1])
        {
            t = L.key[j];
            L.key[j] = L.key[j + 1];
            L.key[j + 1] = t;
        }
}
}

int SelectMinKey(SqList& L, int n)
{
int min = n;
int minkey;//最小值
minkey = L.key[n];
for (int i = n + 1;i <= L.length;i++)
    if (L.key[i]<minkey)
    {
        minkey = L.key[i];
        min = i;
    }
return min;
}

void SelectSort(SqList& L)
{
//对顺序表L作简单选择排序
int j;
int t;
for (int i = 1;i <= L.length;i++)
```

```
{
    j = SelectMinKey(L, i);//在L.key[i]--L.key[L.length]中选择最小的记录并将其地址赋给j
    if (i != j)//交换记录
    {
        t = L.key[i];
        L.key[i] = L.key[j];
        L.key[j] = t;
    }
}
}



int main()
{
SqList s;
cout << "输入元素个数" << endl;
int n;
cin >> n;
s.key = new int[n + 1];
s.length = n;
s.key[0] = 0;
for (int i = 1;i <= n;i++)
{
    cout << "输入第" << i << "个元素" << endl;
    cin >> s.key[i];
}
SqList a = s;
SqList b = s;

BubbleSort(a);
cout << "冒泡排序为：" << endl;
```

```
for (int i = 1;i <= n;i++)
{
    cout << a.key[i] << ' ';
}
cout << "哈希排序输入步长序列数" << endl;
int n1;
cin >> n1;
int *q = new int[n1];
cout <<endl;
cout << "输入步长序列" << endl;
for (int i = 0;i < n1;i++)
{
    cin >> q[i];
}
ShellSort(b, q, n1);
    cout << "哈希排序：" << endl;
    for (int i = 1;i <= n;i++)
    {
        cout << b.key[i] << ' ';
    }
    cout << endl;
    cout << "简单选择排序：" << endl;
    SelectSort(s);
    for (int i = 1;i <= n;i++)
    {
        cout << s.key[i] << ' ';
    }
    return 0;
}
```

5 实验总结：

本次编程总共使用了三种排序方法，而这三种编程方法在一起进行编程时，很容易就让我们对其难易程度更有深刻的了解。首先，对于简单选择排序，思路简单，想查表一样，设置了哨兵，而哨兵的使可以减少对整个验空操作，使程序更加节省空间，用易于进行，其时间空间复杂度与简单插入排序一样，都是O（n^2）,性能不如快速排序。希尔排序是把记录按下标的一定增量分组，对每组使用直接排序算法排序；时间空间复杂度为O(n^1.3).冒泡排序C++里已经了解过，算法简单，每一趟吧最大数沉到底部，最后数据从下到上，从小到大增大。最后，本次实验是数据结构课的最后一次实验，经过数据结构实验课的锻炼，使我们对数据结构有了更深刻的理解，使我对其知识起到了重要的影响，增加了我编程的实践活动，为我将来的进一步学习打下了基础。