**1、题目简介**

**[**问题描述**]**

**编程实现二叉树的建立，先序、中序、后序（递归和非递归方法）、层序遍历。求二叉树的高度、宽度，结点数。判断是否为二叉排序树。**

**#include<iostream>**

**using namespace std;**

**#include<malloc.h>**

**#include<fstream>**

**#define max 100**

**#define add 10**

**typedef struct BiTNode//二叉链表结构体**

**{**

**int data;**

**struct BiTNode \*lchild,\*rchild;//左右孩子指针**

**}BiTNode,\*BiTree;**

**typedef struct QNode//队列**

**{**

**BiTNode \*data;//（int data存数据BITNode\* data存树的指针）**

**struct QNode \*next;//下一个数据的指针**

**}QNode,\*QueuePtr;**

**typedef struct**

**{**

**QueuePtr front;//队列头**

**QueuePtr rear;//队列尾**

**}LinkQueue;**

**typedef struct//栈的结构体**

**{**

**BiTNode \* \*base;//栈底指针**

**BiTNode \* \*top;//栈顶指针**

**int stacksize;//栈的大小**

**}Sqstack;**

**void Init\_Queue(LinkQueue &s);//创建队列**

**void EnQueue(LinkQueue &s,BiTNode \*e);//入队**

**void DeQueue(LinkQueue &s,BiTNode \* &e);//出队**

**int Empty\_Queue(LinkQueue s);//判断队是否为空**

**void Init\_Sqstack(Sqstack &s);//构建栈**

**void Push(Sqstack &s,BiTNode \*e);//入栈**

**void Pop(Sqstack &s,BiTNode\* &e);//出栈**

**int Empty\_stack(Sqstack s);//判断栈是否为空**

**void Gottop\_stack(Sqstack s,BiTNode\* &e);//得到栈顶元素**

**void Init\_BiTree(BiTree &T);//创建二叉树**

**void Creat\_BiTree(BiTree &T);//创二叉树建数据**

**void Clear\_BiTree(BiTree &T);//清空树**

**bool Empty\_BiTree(BiTree T);//判断树是否为空**

**int Max(int x,int y);//简易的两个数比较大小**

**int Depth\_BiTree(BiTree T);//树的深度或者高度**

**void Preorder\_BiTree(BiTree T);//先序遍历**

**void Preorder\_\_BiTree(BiTree T);//先序遍历非递归**

**void Inorder\_BiTree(BiTree T);//中序遍历**

**void Inorder\_\_BiTree(BiTree T);//中序遍历非递归**

**void Postorder\_BiTree(BiTree T);//后序遍历**

**void Postorder\_\_BiTree(BiTree T);//后序遍历非递归**

**void div\_BiTree(BiTree T);//层次遍历**

**int Leaf\_BiTree(BiTree T);//求叶子数即宽度**

**int Paixu\_BiTree(BiTree T);//判断是否为二叉排序树**

**void CreatWen\_BiTree();//创建存树文件**

**void CreatHWen\_BiTree();//从文件传值给树**

**void Init\_Queue(LinkQueue &s)//创建队列**

**{**

**s.front=s.rear=(QueuePtr)malloc(sizeof(QNode));**

**if(!s.front)exit(0);**

**s.front->next=NULL;**

**}**

**void EnQueue(LinkQueue &s,BiTNode \*e)//入队**

**{**

**QueuePtr p;**

**p=(QueuePtr)malloc(sizeof(QNode));//开辟一个对应空间存数据**

**if(!p)exit(0);**

**p->data=e;**

**p->next=NULL;**

**s.rear->next=p;//可以看出队头为空，队尾不空**

**s.rear=p;**

**}**

**void DeQueue(LinkQueue &s,BiTNode \* &e)//出队**

**{**

**if(s.rear==s.front)exit(0);**

**QueuePtr p;**

**p=s.front->next;**

**e=p->data;**

**s.front->next=p->next;**

**if(s.rear==p)s.rear=s.front;**

**free(p);//删除空间**

**}**

**int Empty\_Queue(LinkQueue s)//判断队是否为空**

**{**

**if(s.rear==s.front)**

**return 1;**

**else**

**return 0;**

**}**

**void Init\_Sqstack(Sqstack &s)//构建栈**

**{**

**s.base=(BiTNode \*\*)malloc(max\*sizeof(BiTNode\*));**

**if(!s.base)exit(0);//如果失败，关闭**

**s.top=s.base;**

**s.stacksize=max;**

**}**

**void Push(Sqstack &s,BiTNode \*e)//入栈**

**{//s.top-s.base为栈的长度**

**if(s.top-s.base>=s.stacksize)//如果栈满**

**{**

**s.base=(BiTNode \*\*) realloc (s.base,(s.stacksize+add)\*sizeof( BiTNode\*));**

**if( !s.base)exit(0);**

**s.top=s.base+s.stacksize;**

**s.stacksize+=add;**

**}**

**\*s.top=e;//栈顶赋值**

**s.top++;//栈顶指针下移**

**}**

**void Pop(Sqstack &s,BiTNode\* &e)//出栈**

**{**

**if(s.base==s.top)exit(0);//如果栈空，关闭**

**s.top--;//因为栈顶为空，所以栈顶指针下移即出栈**

**e=\*s.top;**

**//e 赋值 栈当前值 即出栈的数据或者指针**

**}**

**int Empty\_stack(Sqstack s)//判断栈是否为空**

**{**

**if(s.base==s.top)**

**return 1;**

**else**

**return 0;**

**}**

**void Gottop\_stack(Sqstack s,BiTNode\* &e)//得到栈顶元素**

**{**

**if(s.base==s.top)exit(0);**

**s.top--;**

**//只需栈顶元素下移**

**e=\*s.top;**

**}**

**void Init\_BiTree(BiTree &T)//创建二叉树**

**{**

**T=(BiTNode \*)malloc(sizeof(BiTNode));**

**if(!T)exit(0);**

**T->lchild=T->rchild=NULL;//左右孩子置空**

**}**

**void Creat\_BiTree(BiTree &T)//创二叉树建数据**

**{//根据先序遍历规则来建树**

**int e;**

**fstream file,file1;**

**BiTNode f;**

**file.open("tree1.dat",ios::in|ios::binary);**

**if(file.fail())**

**{**

**cout<<"file open error!"<<endl;**

**exit(0);**

**}**

**file1.open("tree2.dat",ios::out|ios::binary);**

**if(file1.fail())**

**{**

**cout<<"file1 open error!"<<endl;**

**exit(0);**

**}**

**file.read((char \*)&f,sizeof(f));**

**e=f.data;**

**file.read((char \*)&f,sizeof(f));**

**while(!file.eof())**

**{**

**file1.write((char \*)&f,sizeof(f));**

**file.read((char \*)&f,sizeof(f));**

**}**

**file.close();**

**file1.close();**

**file.open("tree1.dat",ios::out|ios::binary);**

**if(file.fail())**

**{**

**cout<<"file open error!"<<endl;**

**exit(0);**

**}**

**file1.open("tree2.dat",ios::in|ios::binary);**

**if(file1.fail())**

**{**

**cout<<"file1 open error!"<<endl;**

**exit(0);**

**}**

**file1.read((char \*)&f,sizeof(f));**

**while(!file1.eof())**

**{**

**file.write((char \*)&f,sizeof(f));**

**file1.read((char \*)&f,sizeof(f));**

**}**

**file.close();**

**file1.close();**

**cout<<endl;**

**file.open("tree1.dat",ios::in|ios::binary);**

**if(file.fail())**

**{**

**cout<<"file open error!"<<endl;**

**exit(0);**

**}**

**file.read((char \*)&f,sizeof(f));**

**while(!file.eof())**

**{**

**cout<<f.data<<" ";**

**file.read((char \*)&f,sizeof(f));**

**}**

**file.close();**

**cout<<"e:"<<e<<" ";**

**if(e==0)**

**{**

**T=NULL;**

**}**

**else**

**{**

**T=(BiTNode \*)malloc(sizeof(BiTNode));**

**if(!T)exit(0);**

**T->data=e;**

**Creat\_BiTree(T->lchild);**

**Creat\_BiTree(T->rchild);**

**}**

**}**

**void Clear\_BiTree(BiTree &T)//清空树**

**{**

**if(T)**

**{**

**Clear\_BiTree(T->lchild);**

**Clear\_BiTree(T->rchild);**

**free(T);**

**T=NULL;**

**}**

**}**

**bool Empty\_BiTree(BiTree T)//判断树是否为空**

**{**

**if(!T)**

**return false;**

**}**

**int Max(int x,int y)//简易的两个数比较大小**

**{**

**if(x>y)**

**return x;**

**else**

**return y;**

**}**

**int Depth\_BiTree(BiTree T)//树的深度或者高度**

**{**

**if(T==NULL)**

**return 0;**

**else**

**return(Max(Depth\_BiTree(T->lchild),Depth\_BiTree(T->rchild))+1);**

**}**

**void Preorder\_BiTree(BiTree T)//先序遍历**

**{**

**if(T)**

**{**

**cout<<T->data<<" ";//遍历根**

**Preorder\_BiTree(T->lchild);//遍历左子树**

**Preorder\_BiTree(T->rchild);//遍历右子树**

**}**

**}**

**void Preorder\_\_BiTree(BiTree T)//先序遍历非递归**

**{//用栈实现**

**//思想：根入栈，出栈 右子树 左子树入栈**

**//即保证出栈的顺序是按照根 左子树 右子树。**

**Sqstack s;**

**BiTNode \*p;//树的指针**

**Init\_Sqstack(s);//创建一个栈**

**if(T)**

**Push(s,T);//如果树不空，根入栈**

**while(!Empty\_stack(s))//栈不空为循环条件**

**{**

**Pop(s,p);**

**cout<<p->data<<" ";**

**if(p->rchild)//如果右孩子不空入栈**

**{**

**Push(s,p->rchild);**

**}**

**if(p->lchild)//如果左孩子不空入栈**

**{**

**Push(s,p->lchild);**

**}**

**}**

**}**

**void Inorder\_BiTree(BiTree T)//中序遍历**

**{**

**if(T)**

**{**

**Inorder\_BiTree(T->lchild);//遍历左子树**

**cout<<T->data<<" ";//遍历根**

**Inorder\_BiTree(T->rchild);//遍历 右子树**

**}**

**}**

**void Inorder\_\_BiTree(BiTree T)//中序遍历非递归**

**{**

**//思想：不断的将根 左子树入栈，出栈时，右子树入栈**

**//即保证出栈的顺序是按照左子树 根 右子树。**

**BiTNode \*p=T;**

**Sqstack s;//建栈**

**Init\_Sqstack(s);**

**while(p||!Empty\_stack(s))**

**{**

**if(p)**

**{**

**Push(s,p);//根入栈**

**p=p->lchild;//指针移到左孩子**

**}**

**else**

**{**

**Pop(s,p);//出栈**

**cout<<p->data<<" ";**

**p=p->rchild;//右孩子入栈**

**}**

**}**

**}**

**void Postorder\_BiTree(BiTree T)//后序遍历**

**{**

**if(T)**

**{**

**Postorder\_BiTree(T->lchild);//遍历左子树**

**Postorder\_BiTree(T->rchild);//遍历右子树**

**cout<<T->data<<" ";//遍历根**

**}**

**}**

**void Postorder\_\_BiTree(BiTree T)//后序遍历非递归**

**{**

**//思想：控制出栈的条件是左子树的左右孩子为空，或者是已经访问过**

**BiTNode \*p=T;**

**Sqstack s;**

**Init\_Sqstack(s);//建栈**

**int word=1;**

**if(p->rchild)**

**{**

**Push(s,p->rchild);//右孩子入栈**

**}**

**Push(s,p);//根入栈**

**if(p->lchild)//左孩子入栈**

**{**

**Push(s,p->lchild);**

**}//先存入第一个右孩子和左孩子，根**

**Gottop\_stack(s,p);//得到栈顶元素，即左孩子**

**while(p)//得到左孩子**

**{**

**if(p->rchild&&p->rchild->data!='0')**

**{**

**Push(s,p->rchild);//右孩子入栈**

**}**

**if(p->lchild&&p->lchild->data!='0')**

**{**

**Push(s,p->lchild);//左孩子入栈**

**}**

**//注：根是原来的左孩子已经不需要入栈**

**if((p->lchild==NULL||p->lchild->data=='0')&&(p->rchild==NULL||p->rchild->data=='0'))**

**{**

**Pop(s,p);//出栈**

**cout<<p->data<<" ";**

**T=p;**

**T->data='0';//将树的数据暂时置‘0’**

**}**

**Gottop\_stack(s,p);//得到栈顶元素**

**}**

**}**

**void div\_BiTree(BiTree T)//层次遍历**

**{**

**LinkQueue s;**

**BiTNode \*p=T;**

**Init\_Queue(s);//建队**

**if(T)**

**{**

**EnQueue(s,p);//入队**

**}**

**while(!Empty\_Queue(s))**

**{**

**DeQueue(s,p);//出队**

**cout<<p->data<<" ";**

**if(p->lchild)**

**EnQueue(s,p->lchild);//左孩子入队**

**if(p->rchild)**

**EnQueue(s,p->rchild);//右孩子入队**

**}**

**}**

**int Leaf\_BiTree(BiTree T)//求叶子数即宽度**

**{**

**if(T==NULL)**

**return 0;**

**else**

**{**

**if(T->lchild==NULL&&T->rchild==NULL)**

**return 1;**

**else**

**return(Leaf\_BiTree(T->lchild)+(Leaf\_BiTree(T->rchild)));**

**}**

**}**

**int Paixu\_BiTree(BiTree T)//判断是否为二叉排序树**

**{**

**//思想：通过比较根与左孩子 根与右孩子的情况**

**if(T==NULL)exit(0);**

**else**

**{**

**if(T->lchild)//如果存在左孩子**

**{**

**if(T->data < T->lchild->data)**

**return 0;**

**else**

**{**

**if(T->rchild)**

**{**

**if(T->data<T->rchild->data)**

**{**

**Paixu\_BiTree(T->lchild);**

**Paixu\_BiTree(T->rchild);**

**}**

**else**

**return 0;**

**}**

**else**

**Paixu\_BiTree(T->lchild);**

**}**

**}**

**else//如果不存在左孩子**

**{**

**if(T->rchild)**

**if(T->data>T->rchild->data)**

**{**

**Paixu\_BiTree(T->rchild);**

**}**

**else**

**return 0;**

**}**

**return 1;**

**}**

**}**

**void CreatWen\_BiTree()//创建存树文件**

**{**

**fstream file;**

**file.open("tree.dat",ios::out|ios::binary);**

**if(file.fail())**

**{**

**cout<<"file open error!"<<endl;**

**exit(0);**

**}**

**BiTNode f;**

**int key=1;**

**cout<<"please input the all data(0->NULL -1->end):"<<endl;**

**while(key)**

**{**

**int elem;**

**cin>>elem;**

**f.data=elem;**

**f.lchild=NULL;**

**f.rchild=NULL;**

**if(elem==-1)**

**{**

**key=0;**

**}**

**else**

**file.write((char \*)&f,sizeof(f));**

**}**

**file.close();**

**file.open("tree.dat",ios::in|ios::binary);**

**if(file.fail())**

**{**

**cout<<"file open error!"<<endl;**

**exit(0);**

**}**

**file.read((char \*)&f,sizeof(f));**

**while(!file.eof())**

**{**

**cout<<f.data<<" ";**

**file.read((char \*)&f,sizeof(f));**

**}**

**file.close();**

**cout<<endl;**

**}**

**void CreatHWen\_BiTree()**

**{**

**fstream file,file1;**

**BiTNode f;**

**file.open("tree.dat",ios::in|ios::binary);**

**if(file.fail())**

**{**

**cout<<"file open error!"<<endl;**

**exit(0);**

**}**

**file1.open("tree1.dat",ios::out|ios::binary);**

**if(file1.fail())**

**{**

**cout<<"file1 open error!"<<endl;**

**exit(0);**

**}**

**file.read((char \*)&f,sizeof(f));**

**while(!file.eof())**

**{**

**file1.write((char \*)&f,sizeof(f));**

**file.read((char \*)&f,sizeof(f));**

**}**

**file.close();**

**file1.close();**

**cout<<endl;**

**file.open("tree1.dat",ios::in|ios::binary);**

**if(file.fail())**

**{**

**cout<<"file open error!"<<endl;**

**exit(0);**

**}**

**file.read((char \*)&f,sizeof(f));**

**while(!file.eof())**

**{**

**cout<<f.data<<" ";**

**file.read((char \*)&f,sizeof(f));**

**}**

**file.close();**

**}**

**int main()**

**{**

**int word;**

**cout<<" 二叉树的应用"<<endl;**

**cout<<" 0.创建文件"<<endl;**

**cout<<" 1.构建二叉树 "<<endl;**

**cout<<" 2.求树的高度"<<endl;**

**cout<<" 3.求树的深度"<<endl;**

**cout<<" 4.先序遍历"<<endl;**

**cout<<" 5.中序遍历"<<endl;**

**cout<<" 6.后序遍历"<<endl;**

**cout<<" 7.先序遍历非递归"<<endl;**

**cout<<" 8.中序遍历非递归"<<endl;**

**cout<<" 9.后序遍历非递归"<<endl;**

**cout<<" 10.判断是否为二叉排序树"<<endl;**

**cout<<"please input your choose:";**

**cin>>word;**

**switch(word)**

**{**

**case 0:**

**{**

**system("cls");**

**cout<<" 二叉树的应用"<<endl;**

**cout<<" 创建文件"<<endl;**

**CreatWen\_BiTree();**

**CreatHWen\_BiTree();**

**break;**

**}**

**case 1:**

**{**

**system("cls");**

**cout<<" 二叉树的应用"<<endl;**

**cout<<" 构建二叉树 "<<endl;**

**CreatHWen\_BiTree();**

**BiTree T;**

**Init\_BiTree(T);//构建二叉树**

**Creat\_BiTree(T);//初始化二叉树**

**cout<<endl;**

**Preorder\_BiTree(T);//先序遍历 \*/**

**break;**

**}**

**case 2:**

**{**

**system("cls");**

**cout<<" 二叉树的应用"<<endl;**

**cout<<" 求树的高度"<<endl;**

**CreatHWen\_BiTree();**

**BiTree T;**

**Init\_BiTree(T);//构建二叉树**

**Creat\_BiTree(T);//初始化二叉树**

**cout<<endl;**

**cout<<"Depth:"<<Depth\_BiTree(T)<<endl;**

**break;**

**}**

**case 3:**

**{**

**system("cls");**

**cout<<" 二叉树的应用"<<endl;**

**cout<<" 求树的深度"<<endl;**

**CreatHWen\_BiTree();**

**BiTree T;**

**Init\_BiTree(T);//构建二叉树**

**Creat\_BiTree(T);//初始化二叉树**

**cout<<endl;**

**cout<<"deepth:"<<Leaf\_BiTree(T)<<endl;**

**break;**

**}**

**case 4:**

**{**

**system("cls");**

**cout<<" 二叉树的应用"<<endl;**

**cout<<" 先序遍历"<<endl;**

**CreatHWen\_BiTree();**

**BiTree T;**

**Init\_BiTree(T);//构建二叉树**

**Creat\_BiTree(T);//初始化二叉树**

**cout<<endl;**

**Preorder\_BiTree(T);//先序遍历**

**break;**

**}**

**case 5:**

**{**

**system("cls");**

**cout<<" 二叉树的应用"<<endl;**

**cout<<" 中序遍历"<<endl;**

**CreatHWen\_BiTree();**

**BiTree T;**

**Init\_BiTree(T);//构建二叉树**

**Creat\_BiTree(T);//初始化二叉树**

**cout<<endl;**

**Inorder\_BiTree(T);//中序遍历**

**break;**

**}**

**case 6:**

**{**

**system("cls");**

**cout<<" 二叉树的应用"<<endl;**

**cout<<" 后序遍历"<<endl;**

**CreatHWen\_BiTree();**

**BiTree T;**

**Init\_BiTree(T);//构建二叉树**

**Creat\_BiTree(T);//初始化二叉树**

**cout<<endl;**

**Postorder\_BiTree(T);//后序遍历**

**break;**

**}**

**case 7:**

**{**

**system("cls");**

**cout<<" 二叉树的应用"<<endl;**

**cout<<" 先序遍历非递归"<<endl;**

**CreatHWen\_BiTree();**

**BiTree T;**

**Init\_BiTree(T);//构建二叉树**

**Creat\_BiTree(T);//初始化二叉树**

**cout<<endl;**

**Preorder\_\_BiTree(T);//先序遍历**

**break;**

**}**

**case 8:**

**{**

**system("cls");**

**cout<<" 二叉树的应用"<<endl;**

**cout<<" 中序遍历非递归"<<endl;**

**CreatHWen\_BiTree();**

**BiTree T;**

**Init\_BiTree(T);//构建二叉树**

**Creat\_BiTree(T);//初始化二叉树**

**cout<<endl;**

**Inorder\_\_BiTree(T);//中序遍历**

**break;**

**}**

**case 9:**

**{**

**system("cls");**

**cout<<" 二叉树的应用"<<endl;**

**cout<<" 后序遍历非递归"<<endl;**

**CreatHWen\_BiTree();**

**BiTree T;**

**Init\_BiTree(T);//构建二叉树**

**Creat\_BiTree(T);//初始化二叉树**

**cout<<endl;**

**Postorder\_\_BiTree(T);//后序遍历**

**break;**

**}**

**case 10:**

**{**

**system("cls");**

**cout<<" 二叉树的应用"<<endl;**

**cout<<" 判断是否为二叉排序树"<<endl;**

**CreatHWen\_BiTree();**

**BiTree T;**

**Init\_BiTree(T);//构建二叉树**

**Creat\_BiTree(T);//初始化二叉树**

**cout<<endl;**

**cout<<"1(true)or 0(false):"<<Paixu\_BiTree(T)<<endl;**

**break;**

**}**

**default:**

**cout<<"input data is error!";**

**}**

**return 0;**

1. **题目简介**

**利用随机函数产生10个样本的20000个随机整数（其中之一已经是正序，之一是逆序），利用直接插入排序、希尔排序，冒泡排序、快速排序、选择排序、堆排序，归并排序、基数排序8种排序方法进行排序（结果为由小到大的顺序），并统计每一种排序算法对不同样本所耗费的时间。**

**#include<iostream>**

**using namespace std;**

**#include<malloc.h>**

**#include<ctime>**

**#include<cstdlib>**

**#include<fstream>**

**#include<string>**

**#define max 20001**

**#define min 10**

**typedef struct**

**{**

**int \*elem;**

**int length;**

**int listsize;**

**}SqList;**

**struct QNode**

**{**

**int data;**

**QNode \*next;**

**};**

**typedef struct LinkQueue**

**{**

**QNode \*front;**

**QNode \*rear;**

**}\*Link\_Queue;**

**bool InitQueue(Link\_Queue &Q);**

**void DestroyQueue(Link\_Queue Q);**

**void RadixSort(int \*L,int l\_len);//基数排序**

**void Distribute(QNode \*Node\_head,LinkQueue \*Queue,int i);//分配**

**void Collection(QNode \*Node\_head,LinkQueue \*Queue); //收集**

**int Get\_key(QNode \*p,int i);//得到位值**

**void Traverse(int \*L,int l\_len);**

**void Init\_sq(SqList &L);**

**void Creat\_sq(SqList &L,int n);//对顺序表初始化**

**void Traverse\_sq(SqList L);//遍历所有元素**

**void InsertSort(SqList &L);//简单插入排序**

**void ShellInsert(SqList &L,int dk);//希尔排序**

**void ShellSort(SqList &L,int dlta[],int t);//希尔排序**

**void BubbleSort(SqList &L);//冒泡排序**

**int Partion(SqList &L,int low,int high);//确定L.elem[i]元素的位置**

**void QSort(SqList &L,int low,int high);//快速排序**

**void SelectSort(SqList &L);//简单选择排序**

**void HeapAdjust(SqList &L,int s,int m);//大顶堆**

**void HeadSort(SqList &L);//堆排序**

**void Merge(SqList S,SqList &T,int s,int m,int t);//归并**

**void MergePass(SqList &S,SqList T,int n,int h);//一次归并并排序**

**void MergeSort(SqList &S,SqList T,int n); //归并排序**

**void fuzhi\_wen(SqList &L);//将文件中的文件内容传给顺序表**

**void creat\_wen();//正序文件**

**void creat1\_wen();//逆序文件**

**void creat1\_wen();//逆序文件**

**void Init\_sq(SqList &L)**

**{**

**L.elem=(int \*)malloc(max\*sizeof(int));**

**if(!L.elem)exit(0);**

**L.length=0;**

**L.listsize=max;**

**}**

**void Creat\_sq(SqList &L,int n)//对顺序表初始化**

**{**

**if(n>max)exit(0);**

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

**{**

**cin>>L.elem[i];**

**L.length++;**

**}**

**}**

**void Traverse\_sq(SqList L)//遍历所有元素**

**{**

**if (!L.elem)**

**return;**

**int i=1;**

**while(L.elem[i]&&i<=L.length)**

**{**

**if((i)%100==0)**

**{**

**cout<<endl;**

**}**

**cout<<L.elem[i]<<" ";**

**i++;**

**}**

**cout<<endl;**

**}**

**void InsertSort(SqList &L)//简单插入排序**

**{//在有序的序列里插入未有序的**

**for(int i=1;i<=L.length;i++)**

**{**

**if(L.elem[i]>L.elem[i-1])//如果大于有序的最后一个数，不用排序**

**continue;**

**else**

**{**

**int t=i-1;**

**while(L.elem[i]<L.elem[t])//循环找到待插位置**

**{**

**t--;**

**}**

**t++;//待插位置**

**int temp=L.elem[i];//记录临时**

**for(int j=i;j>t;j--)//后移**

**{**

**L.elem[j]=L.elem[j-1];**

**}**

**L.elem[t]=temp;**

**}**

**}**

**}**

**void ShellInsert(SqList &L,int dk)//希尔排序**

**{**

**for(int i=dk;i<=L.length;i++)//从dk+1开始**

**{**

**if(L.elem[i]<L.elem[i-dk])**

**{**

**int t=L.elem[i];//临时位置**

**int j=i-dk;**

**while(j>=0&&(t<L.elem[j]))//对所有间隔dk的数排序**

**{**

**L.elem[j+dk]=L.elem[j];**

**//记录后移，因为如果放在这个位置，会影响这个位置的希尔排序**

**j-=dk;**

**}**

**L.elem[j+dk]=t;**

**}**

**}**

**}**

**void ShellSort(SqList &L,int dlta[],int t)//希尔排序**

**{**

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

**{**

**ShellInsert(L,dlta[k]);**

**}**

**}**

**void BubbleSort(SqList &L)//冒泡排序**

**{**

**for(int i=0;i<=L.length;i++)**

**for(int j=i+1;j<=L.length;j++)**

**{**

**if(L.elem[i]>L.elem[j])**

**{**

**int t=L.elem[i];**

**L.elem[i]=L.elem[j];**

**L.elem[j]=t;**

**}**

**}**

**}**

**int Partion(SqList &L,int low,int high)//确定L.elem[i]元素的位置**

**{**

**int i=low;**

**int j=high;**

**int t=L.elem[i];**

**while(i<j)**

**{**

**while(L.elem[j]>=t&&i<j)**

**{**

**j--;**

**}**

**L.elem[i]=L.elem[j];**

**while(L.elem[i]<=t&&i<j)**

**{**

**i++;**

**}**

**L.elem[j]=L.elem[i];**

**}**

**L.elem[i]=t;**

**return i;**

**}**

**void QSort(SqList &L,int low,int high)//快速排序**

**{**

**if(low<high)**

**{**

**int key=Partion(L,low,high);**

**QSort(L,low,key-1);**

**QSort(L,key+1,high);**

**}**

**}**

**void SelectSort(SqList &L)//简单选择排序**

**//在i之后选择一个最小的数置于i位置**

**{**

**for(int i=0;i<=L.length-1;i++)**

**{**

**int t=L.elem[i];**

**for(int j=i+1;j<=L.length;j++)**

**{**

**if(L.elem[j]<t)**

**{**

**int t1=L.elem[j];**

**L.elem[j]=t;**

**t=t1;**

**}**

**}**

**L.elem[i]=t;**

**}**

**}**

**void HeapAdjust(SqList &L,int s,int m)//大顶堆**

**{**

**int t=L.elem[s];**

**for(int j=2\*s;j<=m;j\*=2)//从非叶子节点向下搜索最小的值置于根**

**{**

**if(j<m&&(L.elem[j]<L.elem[j+1]))**

**j++;//比较左孩子右孩子，j指向较大者**

**if(t>L.elem[j])**

**break;//如果t比左右孩子都大**

**L.elem[s]=L.elem[j];//将s位置放置左右孩子的最大的值**

**s=j;//s下移到j 位置**

**}**

**L.elem[s]=t;//这里的s位置已变**

**}**

**void HeadSort(SqList &L)//堆排序**

**{**

**for(int i=L.length/2;i>0;i--)**

**HeapAdjust(L,i,L.length);**

**for(int j=L.length;j>1;j--)**

**{**

**int t=L.elem[1];**

**L.elem[1]=L.elem[j];**

**L.elem[j]=t;//将最大的值放在最后一个位置**

**HeapAdjust(L,1,j-1);//去除最后一个值重新创建大顶堆**

**}**

**}**

**void Merge(SqList S,SqList &T,int s,int m,int t)//归并**

**{//将S中的数据归并并排序T中 s为第一个有序序列起点 m+1为第二个有序序列起点**

**int i=s,j=m+1,k=s;//记录位置**

**while(i<=m&&j<=t)**

**{**

**if(S.elem[i]<=S.elem[j])**

**T.elem[k++]=S.elem[i++];**

**else**

**T.elem[k++]=S.elem[j++];**

**}**

**if(i<=m)**

**while(i<=m)**

**T.elem[k++]=S.elem[i++];**

**else while(j<=t)**

**T.elem[k++]=S.elem[j++];**

**}**

**void MergePass(SqList &S,SqList T,int n,int h)//一次归并并排序**

**{**

**int i=1;**

**while(i<=n-2\*h+1)//将序列分成h为单位的序列**

**{**

**Merge(S,T,i,i+h-1,i+2\*h-1);//归并0-h h-2h到**

**i+=2\*h;//每2h归并一次**

**}**

**if(i<n-h+1)Merge(S,T,i,i+h-1,n);//最后一个序列不是h 为单位的序列且小于h**

**else//剩下一个子序列没有归并**

**{**

**for(int k=i;k<=n;k++)**

**{**

**T.elem[k]=S.elem[k];**

**}**

**}**

**}**

**void MergeSort(SqList &S,SqList T,int n)**

**{**

**int h=1;//初始子序列长度为1 n为序列长度**

**while(h<n)**

**{**

**MergePass(S,T,n,h);//h的长度不断变化**

**h=2\*h;**

**MergePass(T,S,n,h);**

**h=2\*h;**

**}**

**}**

**#define key 5**

**bool InitQueue(LinkQueue \*Q)**

**{**

**Q->front=new QNode;**

**if(!(Q->front))**

**return false;**

**Q->front->next=NULL;**

**Q->front->data=-1;**

**Q->rear=Q->front;**

**return true;**

**}**

**void DestroyQueue(LinkQueue \*Q)**

**{**

**Q->rear=NULL;**

**delete Q->front;**

**Q->front=NULL;**

**}**

**void RadixSort(SqList &L,int l\_len)//基数排序**

**{**

**LinkQueue \*Queue=new LinkQueue[10];**

**QNode \*Node\_head=new QNode,\*p,\*pre\_p;**

**int i;**

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

**InitQueue(Queue+i);//构造桶**

**pre\_p=Node\_head;**

**for(i=1;i<=l\_len;i++)//顺序表转链表**

**{**

**p=new QNode;**

**p->data=L.elem[i];**

**pre\_p->next=p;**

**pre\_p=p;**

**}**

**pre\_p->next=NULL;**

**for(i=1;i<=key;i++)//分配、收集**

**{**

**Distribute(Node\_head,Queue,i);**

**Collection(Node\_head,Queue);**

**}**

**p=Node\_head->next;**

**for(i=1;i<=l\_len;i++,p=p->next)//链表信息写入顺序表**

**L.elem[i]=p->data;**

**pre\_p=Node\_head;**

**while(pre\_p)**

**{**

**p=pre\_p->next;**

**delete pre\_p;**

**pre\_p=p;**

**}**

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

**DestroyQueue(Queue+i);**

**delete []Queue;**

**}**

**void Distribute(QNode \*Node\_head,Link\_Queue Queue,int i)//分配**

**{**

**QNode \*p=Node\_head->next;**

**int t;**

**while(p)**

**{**

**t=Get\_key(p,i);**

**Queue[t].rear->next=p;**

**Queue[t].rear=p;**

**p=p->next;**

**Queue[t].rear->next=NULL;**

**}**

**Node\_head->next=NULL;**

**}**

**int Get\_key(QNode \*p,int i)//得到位数**

**{**

**int j,s=p->data;**

**for(j=1;j<i;j++)**

**s/=10;**

**return s%10;**

**}**

**void Collection(QNode \*Node\_head,Link\_Queue Queue)//收集**

**{**

**int i;**

**QNode \*p=Node\_head;**

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

**if(Queue[i].front->next)**

**{**

**p->next=Queue[i].front->next;**

**p=Queue[i].rear;**

**Queue[i].front->next=NULL;**

**Queue[i].rear=Queue[i].front;**

**}**

**}**

**void fuzhi\_wen(SqList &L)//将文件中的文件内容传给顺序表**

**{**

**fstream file;**

**file.open("paixu.txt",ios::in);**

**if(file.fail())**

**{**

**cout<<"file open is error!"<<endl;**

**exit(0);**

**}**

**int t,i=1;**

**file>>t;**

**L.length=0;**

**while(!file.eof())**

**{**

**L.elem[i]=t;**

**L.length++;**

**i++;**

**file>>t;**

**}**

**file.close();**

**}**

**void creat\_wen()//正序文件**

**{**

**fstream file;**

**file.open("paixu.txt",ios::out);**

**if(file.fail())**

**{**

**cout<<"file open is error!"<<endl;**

**exit(0);**

**}**

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

**{**

**file<<i<<" ";**

**//cout<<i<<" ";**

**}**

**file.close();**

**// fstream file;**

**/\* file.open("paixu.txt",ios::in);**

**if(file.fail())**

**{**

**cout<<"file open is error!"<<endl;**

**exit(0);**

**}**

**int t;**

**file>>t;**

**while(!file.eof())**

**{**

**cout<<t<<" ";**

**file>>t;**

**}**

**file.close();\*\*/**

**}**

**void creat1\_wen()//逆序文件**

**{**

**fstream file;**

**file.open("paixu.txt",ios::out);**

**if(file.fail())**

**{**

**cout<<"file open is error!"<<endl;**

**exit(0);**

**}**

**int t=max-1;**

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

**{**

**file<<t<<" ";**

**t--;**

**}**

**file.close();**

**}**

**void creat2\_wen()//乱序文件**

**{**

**fstream file;**

**file.open("paixu.txt",ios::out);**

**if(file.fail())**

**{**

**cout<<"file open is error!"<<endl;**

**exit(0);**

**}**

**int t;**

**srand((unsigned)time(NULL));**

**t=rand()%20000;**

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

**{**

**file<<t<<" ";**

**t=rand()%20000;**

**}**

**file.close();**

**}**

**int main()**

**{//直接插入排序、希尔排序，冒泡排序、快速排序、选择排序、堆排序，归并排序、基数排序**

**cout<<" 排序算法比较"<<endl;**

**cout<<" 0.创建文件"<<endl;**

**cout<<" 1.直接插入排序"<<endl;**

**cout<<" 2.希尔排序"<<endl;**

**cout<<" 3.冒泡排序"<<endl;**

**cout<<" 4.快速排序"<<endl;**

**cout<<" 5.选择排序"<<endl;**

**cout<<" 6.堆排序"<<endl;**

**cout<<" 7.归并排序"<<endl;**

**cout<<" 8.基数排序"<<endl;**

**cout<<" 9.比较"<<endl;**

**int chioce;**

**cout<<"please int your choice:";**

**cin>>chioce;**

**switch(chioce)**

**{**

**case 0:**

**{**

**system("cls");**

**cout<<" 排序算法比较"<<endl;**

**cout<<" 创建文件"<<endl;**

**cout<<" 1.正序"<<endl;**

**cout<<" 2.乱序"<<endl;**

**cout<<" 3.逆序"<<endl;**

**int c;**

**cout<<"please input your choice:";**

**cin>>c;**

**switch(c)**

**{**

**case 1:**

**creat\_wen();**

**break;**

**case 2:**

**creat2\_wen();**

**break;**

**case 3:**

**creat1\_wen();**

**break;**

**default:**

**cout<<"chioce input is error!"<<endl;**

**break;**

**}**

**/\* SqList L;**

**Init\_sq(L);**

**fuzhi\_wen(L);**

**Traverse\_sq(L);\*/**

**break;**

**}**

**case 1:**

**{**

**system("cls");**

**cout<<" 排序算法比较"<<endl;**

**cout<<" 直接插入排序"<<endl;**

**SqList L;**

**Init\_sq(L);**

**fuzhi\_wen(L);**

**double cost\_time;**

**clock\_t start,end;**

**start=clock();**

**InsertSort(L);**

**end=clock();**

**cost\_time=(double) (end-start)/(CLOCKS\_PER\_SEC);**

**Traverse\_sq(L);**

**cout<<endl;**

**cout<<"time:"<<cost\_time<<endl;**

**break;**

**}**

**case 2:**

**{**

**system("cls");**

**cout<<" 排序算法比较"<<endl;**

**cout<<" 希尔排序"<<endl;**

**SqList L;**

**Init\_sq(L);**

**int a[3]={5,3,1};**

**fuzhi\_wen(L);**

**double cost\_time;**

**clock\_t start,end;**

**start=clock();**

**ShellSort(L,a,3);**

**end=clock();**

**cost\_time=(double) (end-start)/(CLOCKS\_PER\_SEC);**

**Traverse\_sq(L);**

**cout<<"time:"<<cost\_time<<endl;**

**break;**

**}**

**case 3:**

**{**

**system("cls");**

**cout<<" 排序算法比较"<<endl;**

**cout<<" 冒泡排序"<<endl;**

**SqList L;**

**Init\_sq(L);**

**fuzhi\_wen(L);**

**double cost\_time;**

**clock\_t start,end;**

**start=clock();**

**BubbleSort(L);**

**end=clock();**

**cost\_time=(double) (end-start)/(CLOCKS\_PER\_SEC);**

**Traverse\_sq(L);**

**cout<<"time:"<<cost\_time<<endl;**

**break;**

**}**

**case 4:**

**{**

**system("cls");**

**cout<<" 排序算法比较"<<endl;**

**cout<<" 快速排序"<<endl;**

**SqList L;**

**Init\_sq(L);**

**fuzhi\_wen(L);**

**double cost\_time;**

**clock\_t start,end;**

**start=clock();**

**QSort(L,0,L.length);**

**end=clock();**

**cost\_time=(double) (end-start)/(CLOCKS\_PER\_SEC);**

**Traverse\_sq(L);**

**cout<<"time:"<<cost\_time<<endl;**

**break;**

**}**

**case 5:**

**{**

**system("cls");**

**cout<<" 排序算法比较"<<endl;**

**cout<<" 选择排序"<<endl;**

**SqList L;**

**Init\_sq(L);**

**fuzhi\_wen(L);**

**double cost\_time;**

**clock\_t start,end;**

**start=clock();**

**SelectSort(L);**

**end=clock();**

**cost\_time=(double) (end-start)/(CLOCKS\_PER\_SEC);**

**Traverse\_sq(L);**

**cout<<"time:"<<cost\_time<<endl;**

**break;**

**}**

**case 6:**

**{**

**system("cls");**

**cout<<" 排序算法比较"<<endl;**

**cout<<" 堆排序"<<endl;**

**SqList L;**

**Init\_sq(L);**

**fuzhi\_wen(L);**

**double cost\_time;**

**clock\_t start,end;**

**start=clock();**

**HeadSort(L);**

**end=clock();**

**cost\_time=(double) (end-start)/(CLOCKS\_PER\_SEC);**

**Traverse\_sq(L);**

**cout<<"time:"<<cost\_time<<endl;**

**break;**

**}**

**case 7:**

**{**

**system("cls");**

**cout<<" 排序算法比较"<<endl;**

**cout<<" 归并排序"<<endl;**

**SqList L,L1;**

**Init\_sq(L);**

**Init\_sq(L1);**

**fuzhi\_wen(L);**

**for(int i=1;i<=L.length;i++)**

**{**

**L1.elem[i]=L.elem[i];**

**}**

**L1.length=L.length;**

**double cost\_time;**

**clock\_t start,end;**

**start=clock();**

**MergeSort(L,L1,L.length);**

**end=clock();**

**cost\_time=(double) (end-start)/(CLOCKS\_PER\_SEC);**

**Traverse\_sq(L);**

**cout<<"time:"<<cost\_time<<endl;**

**break;**

**}**

**case 8:**

**{**

**system("cls");**

**cout<<" 排序算法比较"<<endl;**

**cout<<" 基数排序"<<endl;**

**SqList L;**

**Init\_sq(L);**

**fuzhi\_wen(L);**

**double cost\_time;**

**clock\_t start,end;**

**start=clock();**

**RadixSort(L,L.length);**

**end=clock();**

**cost\_time=(double) (end-start)/(CLOCKS\_PER\_SEC);**

**Traverse\_sq(L);**

**cout<<"time:"<<cost\_time<<endl;**

**break;**

**}**

**case 9:**

**{**

**system("cls");**

**cout<<" 排序算法比较"<<endl;**

**cout<<" 比较"<<endl;**

**SqList L,L1;**

**Init\_sq(L);**

**fuzhi\_wen(L);**

**double cost\_time;**

**clock\_t start,end;**

**start=clock();**

**InsertSort(L);**

**end=clock();**

**cost\_time=(double) (end-start)/(CLOCKS\_PER\_SEC);**

**cout<<"InsertSort time:"<<cost\_time<<endl;**

**int a[3]={5,3,1};**

**fuzhi\_wen(L);**

**start=clock();**

**end=clock();**

**cost\_time=(double) (end-start)/(CLOCKS\_PER\_SEC);**

**cout<<"ShellSort time:"<<cost\_time<<endl;**

**fuzhi\_wen(L);**

**start=clock();**

**BubbleSort(L);**

**end=clock();**

**cost\_time=(double) (end-start)/(CLOCKS\_PER\_SEC);**

**cout<<"BubbleSort time:"<<cost\_time<<endl;**

**fuzhi\_wen(L);**

**start=clock();**

**QSort(L,0,L.length);**

**end=clock();**

**cost\_time=(double) (end-start)/(CLOCKS\_PER\_SEC);**

**cout<<"QSort time:"<<cost\_time<<endl;**

**fuzhi\_wen(L);**

**start=clock();**

**SelectSort(L);**

**end=clock();**

**cost\_time=(double) (end-start)/(CLOCKS\_PER\_SEC);**

**cout<<"SelectSort time:"<<cost\_time<<endl;**

**fuzhi\_wen(L);**

**start=clock();**

**HeadSort(L);**

**end=clock();**

**cost\_time=(double) (end-start)/(CLOCKS\_PER\_SEC);**

**cout<<"HeadSort time:"<<cost\_time<<endl;**

**Init\_sq(L1);**

**fuzhi\_wen(L);**

**for(int i=1;i<=L.length;i++)**

**{**

**L1.elem[i]=L.elem[i];**

**}**

**L1.length=L.length;**

**start=clock();**

**MergeSort(L,L1,L.length);**

**end=clock();**

**cost\_time=(double) (end-start)/(CLOCKS\_PER\_SEC);**

**cout<<"MergeSort time:"<<cost\_time<<endl;**

**fuzhi\_wen(L);**

**start=clock();**

**RadixSort(L,L.length);**

**end=clock();**

**cost\_time=(double) (end-start)/(CLOCKS\_PER\_SEC);**

**cout<<"RadixSort time:"<<cost\_time<<endl;**

**break;**

**}**

**default:**

**cout<<"chioce input is error!"<<endl;**

**break;**

**}**

**return 0;**

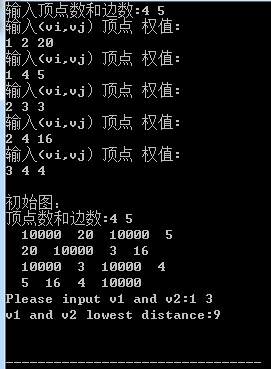
**}**

**一、调试成功程序及说明**

**1、**

**题目：求解图中某一顶点到其他顶点的最短距离。；**

**算法思想：Floyd算法**

****

**// 求解图中某一顶点到其他顶点的最短距离。；**

**#include<iostream>**

**using namespace std;**

**#include<malloc.h>**

**#define Infine 10000**

**#define max 100**

**typedef struct**

**{**

**int \*ver;//顶点**

**int arc[max][max];//边**

**int Ver,Edg;//顶点数和边数**

**}Graph;**

**void Init\_Graph(Graph &g)//创建矩阵-**

**{**

**g.ver=(int \*)malloc(max\*sizeof(int));**

**if(!g.ver)exit(0);**

**int m,n,i,j;**

**cout<<"输入顶点数和边数:";**

**cin>>m>>n;**

**g.Ver=m;//初始化顶点数**

**g.Edg=n;//初始化边数**

**for(i=1;i<=g.Ver;i++)//输入顶点信息标识:"**

**{**

**g.ver[i]=i;**

**}**

**//把矩阵初始化为00**

**for(i=1;i<=m;i++)**

**for(j=1;j<=m;j++)**

**{**

**g.arc[i][j]=Infine;**

**}**

**}**

**void Creat\_Graph(Graph &g)//初始化矩阵**

**{**

**int i,j;**

**//输入有权值的边，其余直接赋值**

**for(int i=1;i<=g.Edg;i++)//输入顶点信息标识:"**

**{**

**int a,a1,weight,word=0;**

**cout<<"输入(vi,vj）顶点 权值:"<<endl;**

**cin>>a>>a1>>weight;////输入顶点 权值**

**for(int s=1;s<=g.Ver;s++)//判断是否顶点是否正确**

**{**

**if(g.ver[s]==a)**

**word++;**

**if(g.ver[s]==a1)**

**word++;**

**}**

**if(word==2)**

**{**

**g.arc[a][a1]=weight;**

**g.arc[a1][a]=weight;//创建有向图图时就不是对称的**

**}**

**else//如果不正确，重复输入**

**{**

**cout<<"input is error!";**

**i--;**

**}**

**}**

**}**

**void Print\_Graph(Graph g)**

**{**

**cout<<"顶点数和边数:"<<g.Ver<<" "<<g.Edg<<endl;**

**for(int i=1;i<=g.Ver;i++)**

**{**

**for(int s=1;s<=g.Ver;s++)**

**{**

**cout<<" "<<g.arc[i][s];**

**}**

**cout<<endl;**

**}**

**}**

**void Floyed\_low(Graph g,int v1,int v2)**

**{**

**for(int s=1;s<=g.Ver;s++)**

**{**

**for(int s1=1;s1<=g.Ver;s1++)**

**{**

**for(int s2=1;s2<=g.Ver;s2++)**

**{**

**if(g.arc[s1][s]+g.arc[s][s2]<g.arc[s1][s2])**

**{**

**g.arc[s1][s2]=g.arc[s1][s]+g.arc[s][s2];**

**}**

**}**

**}**

**}**

**cout<<"v1 and v2 lowest distance:"<<g.arc[v1][v2]<<endl;**

**}**

**int main()**

**{**

**Graph g;**

**Init\_Graph(g);**

**Creat\_Graph(g);**

**cout<<endl;**

**cout<<"初始图："<<endl;**

**Print\_Graph(g);**

**int v1,v2;**

**cout<<"Please input v1 and v2:";**

**cin>>v1>>v2;**

**Floyed\_low(g,v1,v2);**

**cout<<endl;**

**return 0;**

**}**

**2、题目：CSP题目**

**问题描述：小刘承包了很多片麦田，为了灌溉这些麦田，小刘在第一个麦田挖了一口很深的水井，所有的麦田都从这口井来引水灌溉。 为了灌溉，小刘需要建立一些水渠，以连接水井和麦田，小刘也可以利用部分麦田作为“中转站”，利用水渠连接不同的麦田，这样只要一片麦田能被灌溉，则与其连接的麦田也能被灌溉。现在小刘知道哪些麦田之间可以建设水渠和建设每个水渠所需要的费用（注意不是所有麦田之间都可以建立水渠）。请问灌溉所有麦田最少需要多少费用来修建水渠。**

**输入格式：输入的第一行包含两个正整数n, m，分别表示麦田的片数和小刘可以建立的水渠的数量。麦田使用1, 2, 3, ……依次标号。 接下来m行，每行包含三个整数ai, bi, ci，表示第ai片麦田与第bi片麦田之间可以建立一条水渠，所需要的费用为ci。**

**输出格式：输出一个整数，表示灌溉所有麦田所需要的最小费用，及水渠连接说明。**

**问题分析：这个问题可以用最小生成树算法实现。**

**输入样例:**

**4 4**

**1 2 1**

**2 3 4**

**2 4 2**

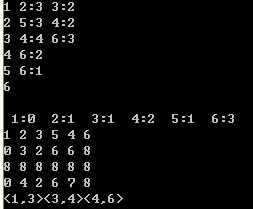
**3 4 3**

**输出样例:**

**6 建立以下3条水渠：麦田1与麦田2、麦田2与麦田4、麦田4与麦田3。**

**算法思想：最小生成树**

**运行结果：**

****

**附源程序。**

**#include<iostream>**

**using namespace std;**

**#include<malloc.h>**

**#define Infine 10000**

**#define max 100**

**typedef struct**

**{**

**int \*ver;//顶点**

**int arc[max][max];//边**

**int Ver,Edg;//顶点数和边数**

**}Graph;**

**void Init\_Graph(Graph &g)**

**{**

**g.ver=(int \*)malloc(max\*sizeof(int));**

**if(!g.ver)exit(0);**

**int m,n,i,j;**

**cout<<"输入顶点数和边数:";**

**cin>>m>>n;**

**g.Ver=m;//初始化顶点数**

**g.Edg=n;//初始化边数**

**for(i=1;i<=g.Ver;i++)//输入顶点信息标识:"**

**{**

**g.ver[i]=i;**

**}**

**//把矩阵初始化为00**

**for(i=1;i<=m;i++)**

**for(j=1;j<=m;j++)**

**{**

**g.arc[i][j]=Infine;**

**}**

**}**

**void Creat\_Graph(Graph &g)**

**{**

**int i,j;**

**//输入有权值的边，其余直接赋值**

**for(int i=1;i<=g.Edg;i++)//输入顶点信息标识:"**

**{**

**int a,a1,weight,word=0;**

**cout<<"输入(vi,vj）顶点 权值:"<<endl;**

**cin>>a>>a1>>weight;////输入顶点 权值**

**for(int s=1;s<=g.Ver;s++)//判断是否顶点是否正确**

**{**

**if(g.ver[s]==a)**

**word++;**

**if(g.ver[s]==a1)**

**word++;**

**}**

**if(word==2)**

**{**

**g.arc[a][a1]=weight;**

**g.arc[a1][a]=weight;**

**}**

**else//如果不正确，重复输入**

**{**

**cout<<"input is error!";**

**i--;**

**}**

**}**

**}**

**void Print\_Graph(Graph g)**

**{**

**cout<<"顶点数和边数:"<<g.Ver<<" "<<g.Edg<<endl;**

**for(int i=1;i<=g.Ver;i++)**

**{**

**for(int s=1;s<=g.Ver;s++)**

**{**

**cout<<" "<<g.arc[i][s];**

**}**

**cout<<endl;**

**}**

**}**

**#define STACK\_INIT\_SIZE 100**

**#define STACKINCREAT 10**

**typedef struct**

**{**

**int \*base;**

**int \*top;**

**int stacksize;**

**}SqStack;**

**void InitStack(SqStack &s)//构造空栈和赋值**

**{**

**s.base=(int \*)malloc(STACK\_INIT\_SIZE\*sizeof(int));**

**if(!s.base)exit(0);**

**s.top=s.base;**

**s.stacksize=STACK\_INIT\_SIZE;**

**}**

**void Print\_Stack(SqStack s)//打印栈**

**{**

**cout<<"the SqStack element:";**

**cout<<s.top-s.base<<endl;**

**SqStack s1;**

**s1.base=s.base;**

**for(int i=0; i<s.top-s.base; i++)//元素的长度length=s.top-s.base**

**{**

**cout<<\*s1.base++<<" ";**

**}**

**}**

**void Push(SqStack &s,int e)//添加一个元素**

**{**

**if(s.top-s.base>=s.stacksize)**

**{**

**s.base=(int \*)realloc(s.base,(s.stacksize+STACKINCREAT)\*sizeof(int));**

**if(!s.base)exit(0);**

**s.top=s.base+s.stacksize;**

**s.stacksize+=STACKINCREAT;**

**}**

**\*s.top=e;**

**s.top++;**

**}**

**void Pop(SqStack &s,int &e)//删除一个元素**

**{**

**if(s.base==s.top)exit(0);**

**--s.top;**

**e=\*s.top;**

**}**

**int Empty\_Stack(SqStack s)**

**{**

**if(s.base==s.top)**

**return 1;**

**else**

**return 0;**

**}**

**void Got\_top(SqStack s,int &e)//得到栈顶的元素**

**{**

**if(s.base==s.top)exit(0);**

**e=\*(s.top-1);**

**}**

**void BFS(Graph g)**

**{**

**;**

**for(int i=1;i<=g.Ver;i++)**

**{**

**for(int s=1;s<=g.Ver;s++)**

**{**

**if(g.arc[i][s]!=Infine)**

**cout<<g.arc[i][s]<<" ";**

**}**

**}**

**}**

**void DFS(SqStack &s,Graph g,int data,int \*visit)**

**{**

**cout<<g.ver[data]<<" ";//输出第一个结点**

**visit[data]=0;//标记为0**

**Push(s,data);//入栈**

**int e;**

**Got\_top(s,e);//得到栈顶点元素**

**while(!Empty\_Stack(s))**

**{**

**int word=0;**

**for(int i=1;i<=g.Ver;i++)**

**{**

**if(g.arc[e][i]!=Infine&&visit[i]!=0)//找到它的一个邻接点**

**{**

**DFS(s,g,i,visit);**

**word=1;**

**break;**

**}**

**}**

**if(word==0)**

**{**

**Pop(s,e);**

**Got\_top(s,e);**

**}**

**for(int i=1;i<=g.Ver;i++)//判断是否还有结点没有找到,果还有，继续这个函数**

**{**

**if(visit[i]!=0)**

**{**

**DFS(s,g,i,visit);**

**}**

**}**

**}**

**}**

**void Prim(Graph g,int k)//k 1**

**{**

**int m,n;**

**int visit[g.Ver+1];//访问标志**

**int lowcost[g.Ver+1];//权值**

**int vex[g.Ver+1];//存顶点**

**for(int i=1;i<=g.Ver;i++)**

**{**

**visit[i]=0;**

**lowcost[i]=g.arc[k][i];**

**vex[i]=1;**

**}**

**visit[k]=1;**

**int key=g.Ver-1;//循环次数**

**cout<<"最小生成树生成边加入过程："<<endl;**

**while(key!=0)**

**{**

**int min;**

**int word;**

**for(int i=1;i<=g.Ver;i++)//设一个最小值**

**{**

**if(!visit[i])//未被访问的**

**{**

**min=lowcost[i];**

**word=i;**

**break;**

**}**

**}**

**for(int i=1;i<=g.Ver;i++)//找到权值最小值**

**{**

**if(!visit[i])**

**{**

**if(lowcost[i]<min)**

**{**

**min=lowcost[i];**

**word=i;//记录位置**

**}**

**}**

**}**

**visit[word]=1;//标记**

**key--;//次数减一**

**cout<<"<"<<vex[word]<<","<<word<<">"<<" ";//输出下一个节点**

**for(int j=1;j<=g.Ver;j++)//刷新权值数组**

**{**

**if(!visit[j])**

**{**

**if(g.arc[word][j]<lowcost[j])**

**{**

**lowcost[j]=g.arc[word][j];//权值改变**

**vex[j]=word;//顶点数组改变**

**}**

**}**

**}**

**}**

**}**

**int main()**

**{**

**Graph g;**

**Init\_Graph(g);**

**Creat\_Graph(g);**

**cout<<endl;**

**cout<<"初始图："<<endl;**

**Print\_Graph(g);**

**int data;**

**cout<<"please input the first number:";**

**cin>>data;**

**cout<<endl;**

**Prim(g,data);**

**/\* SqStack s;**

**InitStack(s);//建栈**

**int visit[max];**

**for(int i=1;i<=g.Ver;i++)**

**{**

**visit[i]=1;**

**}**

**int data;**

**cout<<"please input the first number:";**

**cin>>data;**

**cout<<endl;**

**int word=0;**

**for(int i=1;i<=g.Ver;i++)**

**{**

**if(g.ver[i]==data)**

**{**

**word=1;**

**break;**

**}**

**}**

**if(word==1)**

**{**

**DFS(s,g,data,visit);**

**}**

**else**

**{**

**cout<<"input is error!"<<endl;**

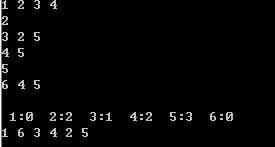
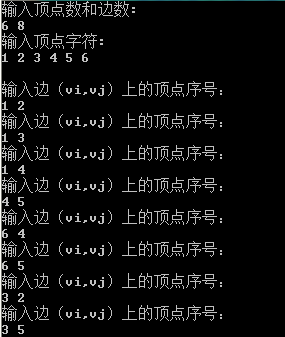
**}\*/**

**return 0;**

**}**

**3、**

**题目：编写拓扑排序算法；**

****

**结果分析：**

**附源程序。**

**#include<iostream>**

**using namespace std;**

**#include<malloc.h>**

**#define MAXVEX 100**

**//边表结点**

**typedef struct EdgeNode**

**{**

**int adjvex;//邻接点域，存储该顶点对应的下标**

**int weight;//用于存储权值，对于非网图可以不需要**

**struct EdgeNode \*next;//链域，指向下一个邻接点**

**}EdgeNode;**

**//顶点表结点**

**typedef struct VertexNode**

**{**

**int data; //顶点域，存储顶点信息**

**EdgeNode \*firstedge; //边表头指针**

**}VertexNode,AdjList[MAXVEX];**

**typedef struct**

**{**

**AdjList adjList;**

**int numVertexes,numEdges;//图中当前顶点数和边数**

**}Graph;**

**void Print\_Graph(Graph \*G)**

**{**

**EdgeNode \*p;**

**for(int i=1;i<=G->numVertexes;i++)**

**{**

**p=G->adjList[i].firstedge;**

**cout<<G->adjList[i].data<<" ";**

**while(p)**

**{**

**cout<<p->adjvex<<" ";**

**p=p->next;**

**}**

**cout<<endl;**

**}**

**}**

**//建立图的邻接表结构**

**//尾插法**

**void Creat\_Graph(Graph \*G)**

**{**

**//输入顶点和边数**

**cout<<"输入顶点数和边数:"<<endl;**

**cin>>G->numVertexes>>G->numEdges;//输入顶点信息，建立顶点表**

**cout<<"输入顶点字符:"<<endl;**

**for(int i=1;i<=G->numVertexes;i++)**

**{**

**cin>>G->adjList[i].data;**

**G->adjList[i].firstedge=NULL;**

**}**

**cout<<endl;**

**//建立边表**

**int i,j;**

**EdgeNode \*p,\*q;**

**EdgeNode \*p1,\*q1;**

**for(int k=0;k<G->numEdges;k++)**

**{**

**cout<<"输入边（vi,vj）上的顶点序号："<<endl;**

**cin>>i>>j;**

**//（单链表中的头插法）**

**p = (EdgeNode\*)malloc(sizeof(EdgeNode));//向内存申请空间**

**//生成边表结点**

**p->adjvex=j;**

**p->next=NULL;**

**if(G->adjList[i].firstedge==NULL)**

**G->adjList[i].firstedge=p;**

**else**

**{**

**p1=G->adjList[i].firstedge;**

**while(p1->next)**

**p1=p1->next;**

**p1->next=p;**

**}**

**/\* q= (EdgeNode\*)malloc(sizeof(EdgeNode));//向内存申请空间**

**//生成边表结点**

**q->adjvex=i;**

**q->next=NULL;**

**if(G->adjList[j].firstedge==NULL)**

**G->adjList[j].firstedge=q;**

**else**

**{**

**q1=G->adjList[j].firstedge;**

**while(q1->next)**

**q1=q1->next;**

**q1->next=q;**

**}\*/**

**}**

**}**

**typedef struct QNode**

**{**

**int data;**

**struct QNode \*next;**

**}QNode,\*QueuePtr;**

**typedef struct**

**{**

**QueuePtr front;//队头指针**

**QueuePtr rear;//队尾指针**

**}LinkQueue;**

**void InitQueue(LinkQueue &Q)**

**{**

**Q.front=Q.rear=(QNode \*)malloc(sizeof(QNode));**

**if(!Q.front)exit(0);**

**Q.front->next=NULL;**

**}**

**int QueueEmpty(LinkQueue Q)**

**{**

**if(Q.rear==Q.front)**

**return 1;**

**else**

**return 0;**

**}**

**void EnQueue(LinkQueue &s,int e)**

**{**

**QueuePtr p;**

**p=(QueuePtr)malloc(sizeof(QNode));**

**if(!p)exit(0);**

**p->data=e;**

**p->next=NULL;**

**s.rear->next=p;**

**s.rear=p;**

**}**

**void DeQueue(LinkQueue &Q,int &e)**

**{**

**if(Q.front==Q.rear)exit(0);**

**QueuePtr s;**

**s=Q.front->next;**

**e=s->data;**

**Q.front->next=s->next;**

**if(Q.rear==s)**

**Q.rear=Q.front;**

**free(s);**

**}**

**void FindIndegree(Graph G,int \*indegree)**

**{**

**EdgeNode \*p;**

**for(int i=1;i<=G.numVertexes;i++)**

**{**

**p=G.adjList[i].firstedge;**

**while(p)**

**{**

**indegree[p->adjvex]++;**

**p=p->next;**

**}**

**}**

**}**

**void Topologicalsort(Graph G)**

**{**

**int indegree[G.numVertexes];**

**for(int i=1;i<=G.numVertexes;i++)**

**indegree[i]=0;**

**FindIndegree(G,indegree);**

**for(int i=1;i<=G.numVertexes;i++)**

**cout<<" "<<i<<":"<<indegree[i]<<" ";**

**cout<<endl;**

**LinkQueue s;**

**InitQueue(s);**

**for(int i=1;i<=G.numVertexes;i++)**

**{**

**if(indegree[i]==0)**

**EnQueue(s,i);**

**}**

**int count=0;**

**while(!QueueEmpty(s))**

**{**

**int e;**

**DeQueue(s,e);**

**cout<<G.adjList[e].data<<" ";**

**count++;**

**EdgeNode \*p;**

**for(p=G.adjList[e].firstedge;p;p=p->next)**

**{**

**int k;**

**k=p->adjvex;indegree[k]--;**

**if(indegree[k]==0)**

**EnQueue(s,k);**

**}**

**}**

**if(count<G.numVertexes)exit(0);**

**}**

**int main()**

**{**

**Graph G;**

**Creat\_Graph(&G);**

**Print\_Graph(&G);**

**cout<<endl;**

**Topologicalsort(G);**

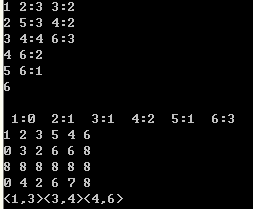
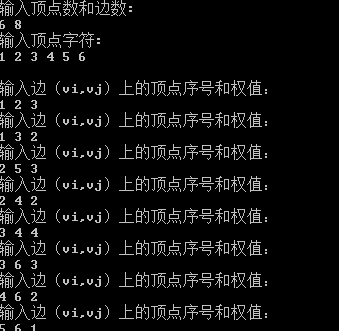
**}**

**4、**

**题目：关键路径练习；**

**算法思想：把书上的代码实现的。**

**运行结果：**

****

**结果分析：**

**附源程序。**

**#include<iostream>**

**using namespace std;**

**#include<malloc.h>**

**#define MAXVEX 100**

**//边表结点**

**typedef struct EdgeNode**

**{**

**int adjvex;//邻接点域，存储该顶点对应的下标**

**int weight;//用于存储权值，对于非网图可以不需要**

**struct EdgeNode \*next;//链域，指向下一个邻接点**

**}EdgeNode;**

**//顶点表结点**

**typedef struct VertexNode**

**{**

**int data; //顶点域，存储顶点信息**

**EdgeNode \*firstedge; //边表头指针**

**}VertexNode,AdjList[MAXVEX];**

**typedef struct**

**{**

**AdjList adjList;**

**int numVertexes,numEdges;//图中当前顶点数和边数**

**}Graph;**

**void Print\_Graph(Graph \*G)**

**{**

**EdgeNode \*p;**

**for(int i=1;i<=G->numVertexes;i++)**

**{**

**p=G->adjList[i].firstedge;**

**cout<<G->adjList[i].data<<" ";**

**while(p)**

**{**

**cout<<p->adjvex<<":";**

**cout<<p->weight<<" ";**

**p=p->next;**

**}**

**cout<<endl;**

**}**

**}**

**/建立图的邻接表结构//尾插法**

**void Creat\_Graph(Graph \*G)**

**{**

**//输入顶点和边数**

**cout<<"输入顶点数和边数:"<<endl;**

**cin>>G->numVertexes>>G->numEdges;**

**//输入顶点信息，建立顶点表**

**cout<<"输入顶点字符:"<<endl;**

**for(int i=1;i<=G->numVertexes;i++)**

**{**

**cin>>G->adjList[i].data;**

**G->adjList[i].firstedge=NULL;**

**}**

**cout<<endl;**

**//建立边表**

**int i,j,weight1;**

**EdgeNode \*p,\*q;**

**EdgeNode \*p1,\*q1;**

**for(int k=0;k<G->numEdges;k++)**

**{**

**cout<<"输入边（vi,vj）上的顶点序号和权值："<<endl;**

**cin>>i>>j>>weight1;**

**//（单链表中的头插法）**

**p = (EdgeNode\*)malloc(sizeof(EdgeNode));//向内存申请空间**

**//生成边表结点**

**p->adjvex=j;**

**p->weight=weight1;**

**p->next=NULL;**

**if(G->adjList[i].firstedge==NULL)**

**G->adjList[i].firstedge=p;**

**else**

**{**

**p1=G->adjList[i].firstedge;**

**while(p1->next)**

**p1=p1->next;**

**p1->next=p;**

**}**

**/\* q= (EdgeNode\*)malloc(sizeof(EdgeNode));//向内存申请空间**

**//生成边表结点**

**q->adjvex=i;**

**q->next=NULL;**

**if(G->adjList[j].firstedge==NULL)**

**G->adjList[j].firstedge=q;**

**else**

**{**

**q1=G->adjList[j].firstedge;**

**while(q1->next)**

**q1=q1->next;**

**q1->next=q;**

**}\*/**

**}**

**}**

**typedef struct QNode**

**{**

**int data;**

**struct QNode \*next;**

**}QNode,\*QueuePtr;**

**typedef struct**

**{**

**QueuePtr front;//队头指针**

**QueuePtr rear;//队尾指针**

**}LinkQueue;**

**void InitQueue(LinkQueue &Q)**

**{**

**Q.front=Q.rear=(QNode \*)malloc(sizeof(QNode));**

**if(!Q.front)exit(0);**

**Q.front->next=NULL;**

**}**

**int QueueEmpty(LinkQueue Q)**

**{**

**if(Q.rear==Q.front)**

**return 1;**

**else**

**return 0;**

**}**

**void EnQueue(LinkQueue &s,int e)**

**{**

**QueuePtr p;**

**p=(QueuePtr)malloc(sizeof(QNode));**

**if(!p)exit(0);**

**p->data=e;**

**p->next=NULL;**

**s.rear->next=p;**

**s.rear=p;**

**}**

**void DeQueue\_last(LinkQueue &Q,int &e)**

**{**

**if(Q.front==Q.rear)exit(0);**

**QueuePtr s;**

**s=Q.front;**

**e=Q.rear->data;**

**while(s)**

**{**

**if(s->next==Q.rear)**

**{**

**Q.rear=s;**

**Q.rear->next=NULL;**

**break;**

**}**

**s=s->next;**

**}**

**}**

**void DeQueue(LinkQueue &Q,int &e)**

**{**

**if(Q.front==Q.rear)exit(0);**

**QueuePtr s;**

**s=Q.front->next;**

**e=s->data;**

**Q.front->next=s->next;**

**if(Q.rear==s)**

**Q.rear=Q.front;**

**free(s);**

**}**

**void FindIndegree(Graph G,int \*indegree)**

**{**

**EdgeNode \*p;**

**for(int i=1;i<=G.numVertexes;i++)**

**{**

**p=G.adjList[i].firstedge;**

**while(p)**

**{**

**indegree[p->adjvex]++;**

**p=p->next;**

**}**

**}**

**}**

**void Topologicalsort(Graph G)**

**{**

**int ve[G.numVertexes+1];**

**int indegree[G.numVertexes+1];**

**for(int i=1;i<=G.numVertexes;i++)**

**{**

**indegree[i]=0;ve[i]=0;**

**}**

**FindIndegree(G,indegree);**

**for(int i=1;i<=G.numVertexes;i++)**

**cout<<" "<<i<<":"<<indegree[i]<<" ";**

**cout<<endl;**

**LinkQueue s,t;**

**InitQueue(s);**

**InitQueue(t);**

**for(int i=1;i<=G.numVertexes;i++)**

**{**

**if(indegree[i]==0)**

**{**

**EnQueue(s,i);**

**}**

**}**

**int count=0;**

**while(!QueueEmpty(s))**

**{**

**int e;**

**DeQueue(s,e);EnQueue(t,e);**

**cout<<G.adjList[e].data<<" ";**

**count++;**

**EdgeNode \*p;**

**for(p=G.adjList[e].firstedge;p;p=p->next)**

**{**

**int k;**

**k=p->adjvex;**

**indegree[k]--;**

**if(indegree[k]==0)**

**{**

**EnQueue(s,k);**

**}**

**if(ve[e]+p->weight>ve[k])**

**ve[k]=ve[e]+p->weight;**

**}**

**}**

**cout<<endl;**

**int vl[G.numVertexes+1];**

**for(int i=1;i<=G.numVertexes;i++)**

**{**

**cout<<ve[i]<<" ";**

**vl[i]=ve[G.numVertexes];**

**}**

**cout<<endl;**

**for(int i=1;i<=G.numVertexes;i++)**

**{**

**cout<<vl[i]<<" ";**

**}**

**while(!QueueEmpty(t))**

**{**

**int e;**

**EdgeNode \*p;**

**for(DeQueue\_last(t,e),p=G.adjList[e].firstedge;p;p=p->next)**

**{**

**int k;**

**// cout<<e<<"\* ";**

**k=p->adjvex;**

**if((vl[k]-p->weight)<vl[e])**

**vl[e]=vl[k]-p->weight;**

**}**

**}**

**cout<<endl;**

**for(int i=1;i<=G.numVertexes;i++)**

**{**

**cout<<vl[i]<<" ";**

**}**

**cout<<endl;**

**for(int j=1;j<=G.numVertexes;j++)**

**{**

**EdgeNode \*q;**

**for(q=G.adjList[j].firstedge;q;q=q->next)**

**{**

**int kk=q->adjvex;**

**int ee=ve[j];**

**int el=vl[kk]-q->weight;**

**if((ee-el)==0)**

**cout<<"<"<<j<<","<<kk<<">";**

**}**

**}**

**if(count<G.numVertexes)exit(0);**

**}**

**int main()**

**{**

**Graph G;**

**Creat\_Graph(&G);**

**Print\_Graph(&G);**

**cout<<endl;**

**Topologicalsort(G);**

**}**

**1．编程实现书P45 ADT Stack 基本操作9个，用顺序存储结构实现；**

**2．编程实现书P59 ADT Queue 基本操作9个，用链式存储结构实现；**

**#include<iostream>**

**using namespace std;**

**#include<malloc.h>**

**#define STACK\_INIT\_SIZE 100**

**#define STACKINCREAT 10**

**typedef struct**

**{**

**int \*base;**

**int \*top;**

**int stacksize;**

**}SqStack;**

**void InitStack(SqStack &s)//构造空栈和赋值**

**{**

**s.base=(int \*)malloc(STACK\_INIT\_SIZE\*sizeof(int));**

**if(!s.base)exit(0);**

**s.top=s.base;**

**s.stacksize=STACK\_INIT\_SIZE;**

**int n;**

**cout<<"The element number:";**

**cin>>n;**

**cout<<"Please input every number:";**

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

**{**

**cin>>\*s.top++;**

**}**

**}**

**void Print\_Stack(SqStack s)//打印栈**

**{**

**cout<<"the SqStack element:";**

**cout<<s.top-s.base<<endl;**

**SqStack s1;**

**s1.base=s.base;**

**for(int i=0; i<s.top-s.base; i++)//元素的长度length=s.top-s.base**

**{**

**cout<<\*s1.base++<<" ";**

**}**

**}**

**void DestroyStack(SqStack &s)//销毁栈**

**{**

**free(s.base);**

**s.top=s.base=NULL;**

**}**

**void ClearStack(SqStack &s)//清空栈**

**{**

**free(s.base);**

**s.top=s.base=NULL;**

**}**

**bool StackEmpty(SqStack &s)//清空栈**

**{**

**if(s.base==NULL)**

**return false;**

**}**

**int StackLength(SqStack s)//返回栈的长度**

**{**

**return (s.top-s.base);**

**}**

**void GetTop(SqStack s,int &e)//得到栈顶的元素**

**{**

**if(s.base==s.top)exit(0);**

**e=\*(s.top-1);**

**}**

**void Push(SqStack &s,int e)//添加一个元素**

**{**

**if(s.top-s.base>=s.stacksize)**

**{**

**s.base=(int \*)realloc(s.base,(s.stacksize+STACKINCREAT)\*sizeof(int));**

**if(!s.base)exit(0);**

**s.top=s.base+s.stacksize;**

**s.stacksize+=STACKINCREAT;**

**}**

**\*s.top=e;**

**s.top++;**

**}**

**void Pop(SqStack &s,int &e)//删除一个元素**

**{**

**if(s.base==s.top)exit(0);**

**--s.top;**

**e=\*s.top;**

**}**

**void visit(int e)**

**{**

**cout<<e<<" ";**

**}**

**void StackTraverse(SqStack s)//对栈调用visit()函数**

**{**

**for(int i=0; i<s.top-s.base; i++)//元素的长度length=s.top-s.base**

**{**

**visit(\*s.base);**

**\*s.base++;**

**}**

**}**

**#include<iostream>**

**using namespace std;**

**#include<malloc.h>**

**typedef struct QNode**

**{**

**int data;**

**struct QNode \*next;**

**}QNode,\*QueuePtr;**

**typedef struct**

**{**

**QueuePtr front;//队头指针**

**QueuePtr rear;//队尾指针**

**}LinkQueue;**

**void InitQueue(LinkQueue &Q)**

**{**

**Q.front=Q.rear=(QNode \*)malloc(sizeof(QNode));**

**if(!Q.front)exit(0);**

**Q.front->next=NULL;**

**int n;**

**cout<<"The element number:";**

**cin>>n;**

**cout<<"Please input every number:";**

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

**{ QNode \*s;**

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

**cin>>s->data;**

**Q.rear->next=s;**

**Q.rear=Q.rear->next;**

**}**

**Q.rear->next=NULL;**

**}**

**void visit (int e)**

**{**

**cout<<e<<" ";**

**}**

**void QueueTraverse(LinkQueue Q)**

**{**

**QueuePtr L;**

**if(Q.front->next!=NULL)**

**{**

**L=Q.front->next;**

**cout<<"output all element:";**

**while(L)**

**{**

**cout<<L->data<<" ";**

**L=L->next;**

**}**

**}**

**}**

**void DestroyQueue(LinkQueue &Q)**

**{**

**Q.front=Q.front->next;**

**while(Q.front)**

**{**

**Q.rear=Q.front->next;**

**free(Q.front);**

**Q.front=Q.rear;**

**}**

**}**

**void ClearQueue(LinkQueue &Q)**

**{**

**Q.front=Q.front->next;**

**while(Q.front)**

**{**

**Q.rear=Q.front->next;**

**free(Q.front);**

**Q.front=Q.rear;**

**}**

**}**

**bool QueueEmpty(LinkQueue Q)**

**{**

**if(Q.front->next==NULL)**

**return ture;**

**else**

**return false;**

**}**

**int QueueLength(LinkQueue Q)**

**{**

**int i=0;**

**Q.front=Q.front->next;**

**while(Q.front)**

**{**

**i++;**

**Q.front=Q.rear;**

**}**

**return i;**

**}**

**void EnQueue(likeQueue &Q,int e)**

**{**

**Queueptr s;**

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

**if(!p)exit(0);**

**s->data=e;**

**Q.rear->next=s;**

**Q.rear=Q.rear->next;**

**Q.rear->next=NULL;**

**}**

**void DeQueue(LinkQueue &Q,int &e)**

**{**

**if(Q.front==Q.rear)exit(0);**

**Queueptr s;**

**s=Q.front->next;**

**e=s->data;**

**Q.front->next=s->next;**

**if(Q.rear==s)**

**Q.rear=Q.front;**

**free(s);**

**}**

**#include<stdio.h>**

**#include<stdlib.h>**

**typedef struct LNode**

**{**

**int data;**

**struct LNode \*next;**

**}LNode,\*LinkList;**

**LinkList Init\_LNode() //链表初始化**

**{**

**LNode \*L;**

**L = (LinkList)malloc(sizeof(LNode));**

**if(L == NULL)**

**{**

**printf("初始化失败!\n");**

**exit(-1);**

**}**

**L->next = NULL;**

**return L;**

**}**

**void Creat\_List1(LNode \*L) //头插法**

**{**

**int i,n,num;**

**LNode \*pnew;**

**printf("请输入要输入的元素个数：n = ");**

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

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

**{**

**printf("请输入第%d个数：",i+1);**

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

**pnew = (LinkList)malloc(sizeof(LNode));**

**pnew->data = num;**

**pnew->next = L->next;**

**L->next = pnew;**

**}**

**}**

**void Creat\_List(LNode \*L) //创建 尾插法**

**{**

**int i,n,num;**

**LNode \*p,\*pnew;**

**p = L;**

**printf("请输入要输入的元素个数：n = ");**

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

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

**{**

**printf("请输入第%d个数：",i+1);**

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

**pnew = (LinkList)malloc(sizeof(LNode));**

**if(pnew == NULL)**

**{**

**printf("初始化失败!\n");**

**exit(-1);**

**}**

**pnew->data = num;**

**p->next = pnew;**

**p = pnew;**

**}**

**p->next =NULL;**

**}**

**void Show\_Linst(LNode \*L) //遍历**

**{**

**LNode \*p;**

**p =L->next;**

**while(p!=NULL)**

**{**

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

**p = p->next;**

**}**

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

**}**

**void Insert\_List(LNode \*L) //插入**

**{**

**int i , n , val;**

**LNode \*p,\*s;**

**p = L;**

**i=1;**

**printf("请输入你要插入的位置：n=");**

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

**printf("请输入你要插入的元素：val=");**

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

**while(p&&i<n)**

**{**

**p = p->next;**

**++i;**

**}**

**if(!p || i>n)**

**printf("无法插入!\n");**

**s = (LinkList)malloc(sizeof(LNode));**

**s->data = val;**

**s->next = p->next;**

**p->next = s;**

**}**

**void Delete\_List(LNode \*L) //删除**

**{**

**int i , n ;**

**LNode \*p,\*s;**

**p = L;**

**i=1;**

**printf("请输入你要删除的位置：n=");**

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

**while(p&&i<n)**

**{**

**p = p->next;**

**++i;**

**}**

**if(!(p->next) || i>n)**

**printf("删除的位置不合法!\n");**

**// p->next =p->next->next;**

**s = p->next;**

**p->next = s->next;**

**free(s);**

**}**

**void Query\_List(LNode \*L) //查询**

**{**

**int i , n ,val;**

**LNode \*p;**

**p = L->next;**

**i=1;**

**printf("请输入你要查询的位置：n=");**

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

**while(p&&i<n)**

**{**

**p = p->next;**

**++i;**

**}**

**if(!(p->next) || i>n)**

**printf("查询的元素不存在!\n");**

**val = p->data;**

**printf("你查询的元素值为：%d\n",val);**

**}**

**void Destroy\_List(LNode \*L) //销毁**

**{**

**LinkList p,q;**

**p = L->next; //p指向第一个结点**

**while(p)**

**{**

**q = p->next;**

**free(p);**

**p = q;**

**}**

**L->next = NULL;**

**}**

**int Show\_Menu() //主菜单**

**{**

**int num;**

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

**printf("\n 【1】创建 【2】插入\n");**

**printf("\n 【3】删除 【4】查询\n");**

**printf("\n 【5】遍历 【6】退出\n");**

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

**printf("请输入你的操作序号：");**

**if(scanf("%d",&num)==0)**

**{**

**printf("请重新输入!\n");**

**}**

**return num;**

**}**