**南京航空航天大学**

**数据结构课程设计报告**

|  |  |
| --- | --- |
| 学生姓名 | 徐宇顺 |
| 学 号 | 031720106 |
| 学 院 | 计算机科学与技术学院 |
| 专 业 | 软件工程 |
| 班 级 | 1617301 |
| 指导教师 | 高航 |

二〇一九年一月

**目 录**

[一、购物网站信息管理 4](#_Toc16524)

[1.1 题目简介 4](#_Toc24720)

[1.2数据结构 4](#_Toc28370)

[1.3 算设法计思想 5](#_Toc9562)

[1.4测试数据和结果 7](#_Toc31152)

[1.5算法时间复杂度 9](#_Toc32400)

[1.6源代码 10](#_Toc27957)

[二、公共钥匙盒 24](#_Toc24745)

[2.1 题目简介 24](#_Toc25235)

[2.2 数据结构 25](#_Toc4239)

[2.3 算法设计思想 25](#_Toc28770)

[2.4测试数据和结果 25](#_Toc30801)

[2.5算法时间复杂度 27](#_Toc4610)

[2.6源代码 27](#_Toc31079)

[三、JSON 33](#_Toc2816)

[3.1 题目简介 33](#_Toc3912)

[3.2 数据结构 33](#_Toc19359)

[3.3 算法设计思想 33](#_Toc25064)

[3.4测试数据和结果 34](#_Toc31772)

[3.5算法时间复杂度 35](#_Toc15368)

[3.6源代码 36](#_Toc8095)

[四、Huffman编码和解码 49](#_Toc24057)

[4.1 题目简介 49](#_Toc11772)

[4.2 数据结构 49](#_Toc3759)

[4.3 算法设计思想 50](#_Toc2661)

[4.4测试数据和结果 50](#_Toc15911)

[4.56算法时间复杂度 51](#_Toc4576)

[4.6源代码 51](#_Toc24227)

[五、行车路线 58](#_Toc8639)

[5.1 题目简介 58](#_Toc6319)

[5.2 数据结构 58](#_Toc4286)

[5.3 算法设计思想 58](#_Toc29013)

[5.4测试数据和结果 59](#_Toc8829)

[5.5算法时间复杂度 60](#_Toc30701)

[5.6源代码 60](#_Toc22481)

[六、排序算法比较 68](#_Toc1727)

[6.1 题目简介 68](#_Toc150)

[6.2 数据结构 68](#_Toc14755)

[6.3 算法设计思想 68](#_Toc23480)

[6.4测试数据和结果 70](#_Toc25757)

[6.5算法时间复杂度 70](#_Toc7219)

[6.6源代码 71](#_Toc24469)

[七、朋友圈 85](#_Toc29741)

[7.1 题目简介 85](#_Toc664)

[7.2 数据结构 85](#_Toc23174)

[7.3 算法设计思想 85](#_Toc8852)

[7.4测试数据和结果 86](#_Toc5111)

[7.5算法时间复杂度 86](#_Toc220)

[7.6源代码 87](#_Toc24053)

[八、社交网络图中结点的“重要性”计算 97](#_Toc26239)

[8.1 题目简介 97](#_Toc3026)

[8.2 数据结构 97](#_Toc13925)

[8.3 算法设计思想 97](#_Toc6513)

[8.4测试数据和结果 98](#_Toc24690)

[8.5算法时间复杂度 109](#_Toc5814)

[8.6源代码 109](#_Toc32518)

[九、魔法优惠劵 115](#_Toc24048)

[9.1 题目简介 115](#_Toc3603)

[9.2 数据结构 115](#_Toc23185)

[9.3 算法设计思想 115](#_Toc1243)

[9.4测试数据和结果 116](#_Toc14056)

[9.5算法时间复杂度 117](#_Toc26546)

[9.6源代码 117](#_Toc13172)

十、 [课程及课设总结 1](#_Toc24048)18

[10.1 完成情况（代码行数） 1](#_Toc3603)18

[10.2 心得体会 1](#_Toc6584)18

# 一、购物网站信息管理

1.1 题目简介

1、购物网站信息管理（必做）（线性表）

[问题描述]

设计一个程序，对商铺信息管理，商铺信息包括：商铺编号，商铺名，信誉度（0-5），（商品名称1，价格1，销量1），（商品名称2，价格2，销量2），（商品名称3，价格3，销量3）…。//商品用顺序表

商品名称包括(毛巾，牙刷，牙膏，肥皂，洗发水，沐浴露等6种以上商品)，每个商铺具有其中事先确定若干商品及价格，由文件输入，销量初始为0。

[基本要求]

（1）建立一个单向链表存储所有商铺信息（至少30个），以编号为序，编号从1开始递增，从文件中读取数据，并能将数据存储在文件。商铺信息结点的数据结构自行设计。

（2）可以增、删商铺。增加商铺，编号自动加一，插入链表尾部；删除商铺，以编号为准，并修改后续结点的编号，保持编号连续性。可增、删商品。

（3）查询某一种商品名称，建立一个双向循环链表，结点信息是包含该商品的所有商铺编号、商铺名、信誉度、商品名称、价格、销量，以信誉度从高至低，并按销量排序，并逐一显示。

（4）购买某一商铺的商品，修改单向链表中商品的信息的销量。

（5）建立一个顺序结构，按商品名Hash分+配地址，存储当前每种商品总销量并输出。

（6）任何的商铺信息变化，实现文件存储

1.2 数据结构

typedef struct goods

{

char Goods\_name[20];

float Price;

int Sales;

}Goods;//商品的信息

typedef struct GoodsNode

{

Goods product;

struct GoodsNode \*next\_goods;

}GoodsNode,\*GoodsList;//构建单个商铺里面的商品链

typedef struct shop

{

int ShopNum;

char ShopName[20];

int reputation;//(0-5)

GoodsList goodslist;//每个商铺含有一个商品链

struct shop \*next\_shop;

}Shop,\*ShopList;

typedef struct search

{

struct search \*front;

struct search \*next;

int ShopNum;

char ShopName[20];

int reputation;

Goods product;

}Search,\*SearchList;//查找的list

typedef struct salenode

{

char GoodsName[20];

int Sales;

}SaleNode;

typedef struct salelist

{

SaleNode \*base;

int length;

int listsize;

}SaleList;

1.3 算法设计思想

建立一个链表，链表中放入商铺的序号，商铺名，以及一个商品链，商品链由一个顺序表构成。增加商铺时，则找到这个链表的末尾，在末尾添加一个新的结点，结点中商铺的编号比前一个结点多一。删除结点时，要将找到商铺对应的编号的结点，然后将该结点用free函数释放空间。

在查询一个商品的时候，建立一个双向链表，用插入法来建立，以商铺的销量排名，在以商品的销量建立一个双向链表。

购买一个商品的时候，直接输入需要购买的商品的店铺，和商品名以及数量，遍历商铺的链表，如果无该商品就输出无该商铺，如果该商铺中无该商品则输出无该商品。

建立一个顺序结构，遍历每个商铺的商品，以商品名作为关键字，每一个商品的时候，则开始检测是否存在，若存在则加上该商品的销量，不存在则新建一个空间，放入该商品名。

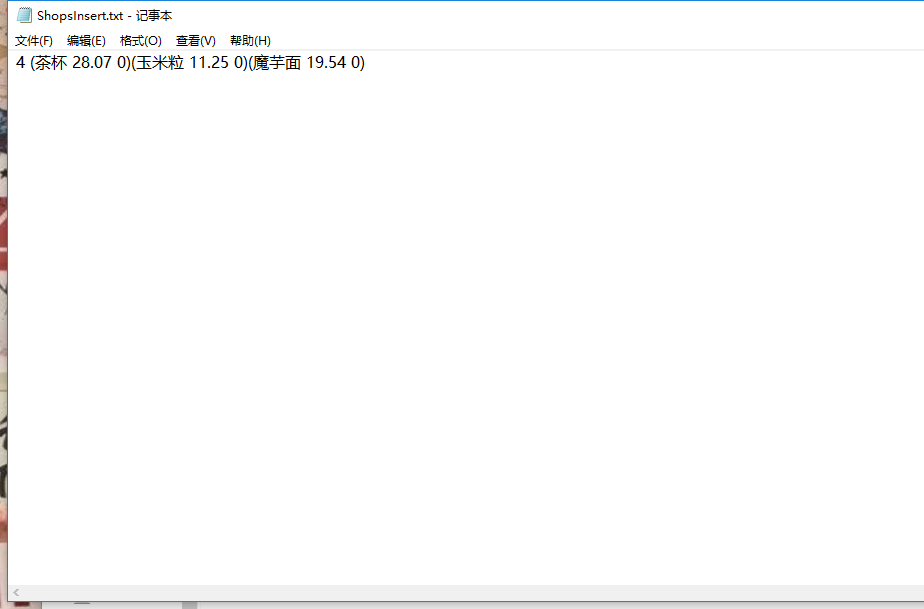
最后每进行一个操作，就输出到所有信息到文本中。

1.4测试数据和结果

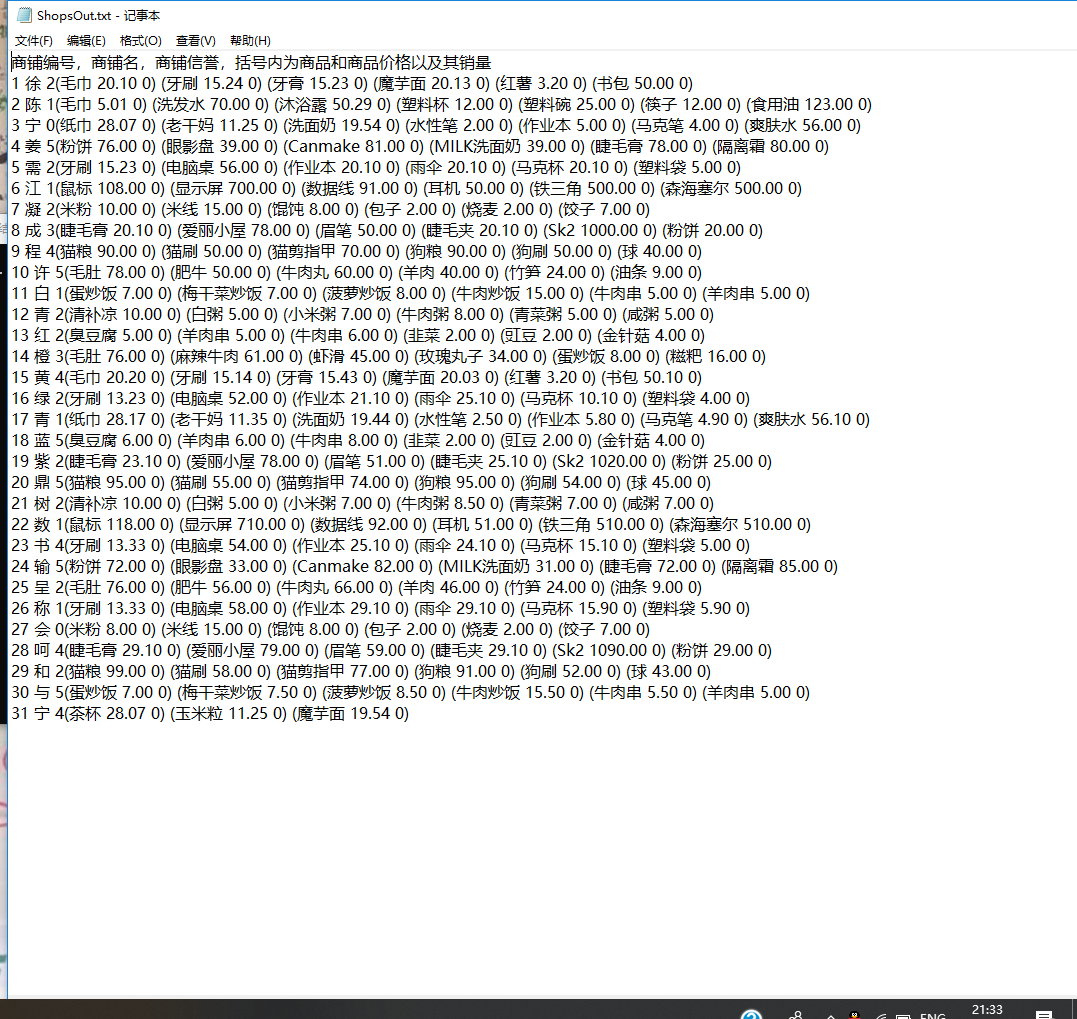




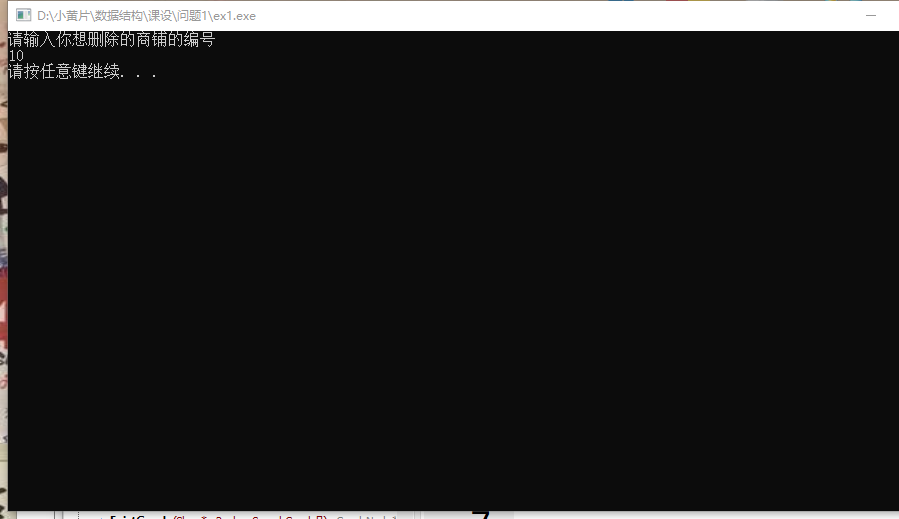
起初建立好所有的商铺链



插入新的商铺结点



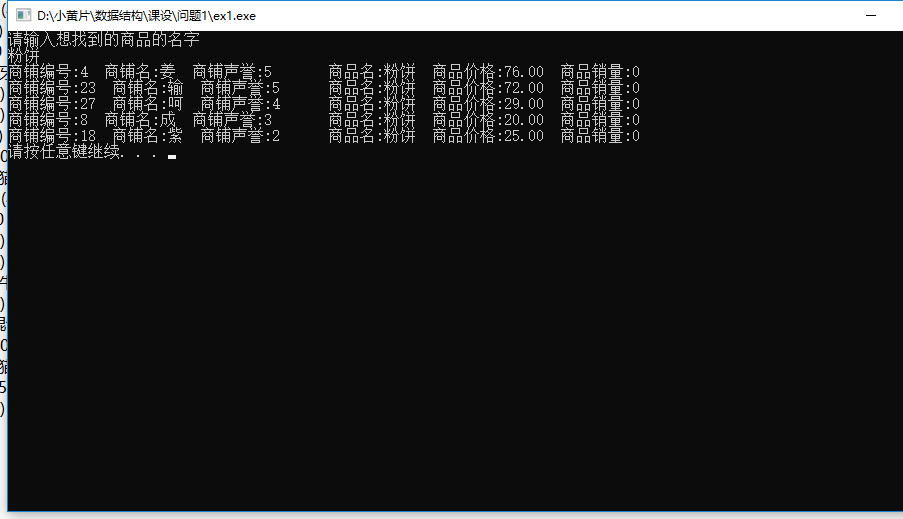
再次输出到ShopsOut.txt的文件中



删除商铺为10的商铺结点



输出到ShopsOut.txt的文件中

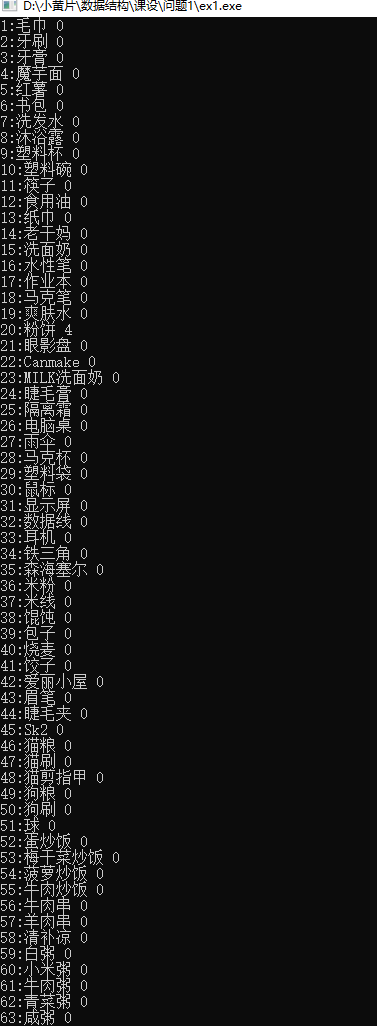


查询一个商品，输出所有商品的信息





购买商品，以及输入销量，在输出到ShopsOut.txt的文件中





输出所有的商品以及商品销量

1.5算法时间复杂度

InitShop(ShopList Shops)

时间复杂度0(n)

InsertSearchList(SearchList searchlist,Shop\* p2,GoodsNode \*q)

时间复杂度O(n)

OutputSearchList(SearchList searchlist)

时间复杂度O(n)

SearchGood(ShopList Shops)

时间复杂度O(n)

printSearchList(SearchList searchlist)

时间复杂度O(n)

BuyGoods(ShopList Shops)

时间复杂度O(n)

InsertSaleList(SaleList \*sale,GoodsNode \*flag)

时间复杂度O(n)

SaleSort(ShopList Shops)

时间复杂度O(n2)

1.6源代码

#include<stdio.h>

#include<malloc.h>

#include<stdlib.h>

#include<string.h>

#define LIST\_INIT\_SIZE 100

#define LISTINCERMENT 10

typedef struct goods

{

char Goods\_name[20];

float Price;

int Sales;

}Goods;

typedef struct GoodsNode

{

Goods product;

struct GoodsNode \*next\_goods;

}GoodsNode,\*GoodsList;//构建单个商铺里面的商品链

typedef struct shop

{

int ShopNum;

char ShopName[20];

int reputation;//(0-5)

GoodsList goodslist;//每个商铺含有一个商品链

struct shop \*next\_shop;

}Shop,\*ShopList;

typedef struct search

{

struct search \*front;

struct search \*next;

int ShopNum;

char ShopName[20];

int reputation;

Goods product;

}Search,\*SearchList;

typedef struct salenode

{

char GoodsName[20];

int Sales;

}SaleNode;

typedef struct salelist

{

SaleNode \*base;

int length;

int listsize;

}SaleList;

ShopList InitShop(ShopList Shops)

{

FILE \*in;

in=fopen("ShopsIn.txt","r");

if(in==NULL)

{

printf("Can't open the file ShopsIn.txt\n");

exit(0);

}

Shop\* p1,\*p2;

Shops=(ShopList)malloc(sizeof(Shop));

p2=Shops;

GoodsNode\* q1,\*q2;

char ch1,ch2;

while(!feof(in))

{

p1=(Shop\*)malloc(sizeof(Shop));

fscanf(in,"%d",&p1->ShopNum);

fscanf(in,"%s",p1->ShopName);

fscanf(in,"%d",&p1->reputation);

p1->goodslist=(GoodsList)malloc(sizeof(GoodsNode));

q2=p1->goodslist;

ch2=fgetc(in);

ch1=fgetc(in);

while(!feof(in)&&ch1=='(')

{

q1=(GoodsNode\*)malloc(sizeof(GoodsNode));

fscanf(in,"%s",q1->product.Goods\_name);

fscanf(in,"%f",&q1->product.Price);

fscanf(in,"%d",&q1->product.Sales);

q2->next\_goods=q1;

q2=q1;

ch2=fgetc(in);

ch1=fgetc(in);

}

q2->next\_goods=NULL;

p2->next\_shop=p1;

p2=p1;

}//建立商店的单链表，以及商铺中的商品链表

p2->next\_shop=NULL;

fclose(in);

return Shops;

}//输入 check

ShopList InsertShop(ShopList Shops)

{

FILE \*insert;

insert=fopen("ShopsInsert.txt","r");

if(insert==NULL)

{

printf("Can't open the file ShopsInsert.txt\n");

exit(0);

}

Shop\* p1,\*p2;

GoodsNode\* q1,\*q2;

char ch1,ch2;

int number;

p1=Shops;

p2=p1->next\_shop;

while(p2)

{

p1=p2;

p2=p1->next\_shop;

}

number=p1->ShopNum;

while(!feof(insert))

{

p2=(Shop\*)malloc(sizeof(Shop));

p2->ShopNum=number+1;

fscanf(insert,"%s",p2->ShopName);

fscanf(insert,"%d",&p2->reputation);

ch2=fgetc(insert);

ch1=fgetc(insert);

p2->goodslist=(GoodsList)malloc(sizeof(GoodsNode));

q2=p2->goodslist;

while(!feof(insert)&&ch1=='(')

{

q1=(GoodsNode\*)malloc(sizeof(GoodsNode));

fscanf(insert,"%s",q1->product.Goods\_name);

fscanf(insert,"%f",&q1->product.Price);

fscanf(insert,"%d",&q1->product.Sales);

q2->next\_goods=q1;

q2=q1;

ch2=fgetc(insert);

ch1=fgetc(insert);

}

printf("\n");

q2->next\_goods=NULL;

number++;

p1->next\_shop=p2;

p1=p2;

}

p1->next\_shop=NULL;

fclose(insert);

return Shops;

}//插入商铺

ShopList DeleteShop(ShopList Shops)

{

int number;

printf("请输入你想删除的商铺的编号\n");

scanf("%d",&number);

Shop\* p1,\*p2,\*s;

p1=Shops;

p2=p1->next\_shop;

while(p2)

{

if(p2->ShopNum==number) break;

p1=p2;

p2=p1->next\_shop;

}

if(p2==NULL){

printf("该商铺不存在\n");

return Shops;

}

else{

s=p2;

p2=p2->next\_shop;

p1->next\_shop=p2;

free(s);

while(p2)

{

p2->ShopNum--;

p1=p2;

p2=p1->next\_shop;

}

}

return Shops;

}//删除商铺

GoodsNode\* ExistGoods(Shop\* p2,char SearchGoods[])

{

GoodsNode \*q1,\*q2;

q1=p2->goodslist;

q2=q1->next\_goods;

while(q2)

{

if(strcmp(q2->product.Goods\_name,SearchGoods)==0)

{

return q2;

}

q1=q2;

q2=q1->next\_goods;

}

return NULL;

}//检查该商铺中有无该商品

SearchList InsertSearchList(SearchList searchlist,Shop\* p2,GoodsNode \*q)

{

Search \*s1,\*s2,\*flag;

s1=searchlist;

s2=s1->next;

while(s2!=searchlist)

{

if(p2->reputation>s2->reputation)

{

break;

}

else if(p2->reputation==s2->reputation)

{

if(q->product.Sales>s2->product.Sales){

break;

}

}

s1=s2;

s2=s1->next;

}

flag=(Search\*)malloc(sizeof(Search));

flag->product=q->product;

flag->reputation=p2->reputation;

strcpy(flag->ShopName,p2->ShopName);

flag->ShopNum=p2->ShopNum;

s1->next=flag;

flag->front=s1;

flag->next=s2;

s2->front=flag;

return searchlist;

}

void OutputSearchList(SearchList searchlist)

{

Search \*s1,\*s2;

s1=searchlist;

s2=s1->next;

while(s2!=searchlist)

{

printf("商铺编号:%d 商铺名:%s 商铺声誉:%d",s2->ShopNum,s2->ShopName,s2->reputation);

printf("\t商品名:%s 商品价格:%.2f 商品销量:%d\n",s2->product.Goods\_name,s2->product.Price,s2->product.Sales);

s1=s2;

s2=s1->next;

}

}

void SearchGood(ShopList Shops)

{

char SearchGoods[20];

printf("请输入想找到的商品的名字\n");

scanf("%s",SearchGoods);

Shop\* p1,\*p2;

GoodsNode\* q;

p1=Shops;

p2=p1->next\_shop;

SearchList searchlist;

searchlist=(SearchList)malloc(sizeof(Search));

searchlist->next=searchlist;

searchlist->front=searchlist;

while(p2)

{

q=ExistGoods(p2,SearchGoods);

if(q)

{

searchlist=InsertSearchList(searchlist,p2,q);

}

p1=p2;

p2=p1->next\_shop;

}

OutputSearchList(searchlist);

}//查询某一种商品名称，建立一个双向循环链表 check

void printSearchList(SearchList searchlist)

{

Search \*s1,\*s2;

int i=1;

s1=searchlist;

s2=s1->next;

while(s2!=searchlist)

{

printf("%d: %d %s %d",i,s2->ShopNum,s2->ShopName,s2->reputation);

printf("(%s %.2f %d)",s2->product.Goods\_name,s2->product.Price,s2->product.Sales);

s1=s2;

s2=s1->next;

printf("\n");

}

}//输出查找商品双向链表的信息

ShopList BuyGoods(ShopList Shops)

{

char BuyShopName[20];

char BuyGoodsName[20];

int count;

printf("请问想在哪个商铺购买商品\n");

scanf("%s",BuyShopName);

printf("请问想购买什么商品以及购买的数量\n");

scanf("%s%d",BuyGoodsName,&count);

Shop \*p1,\*p2;

GoodsNode \*q1,\*q2;

p1=Shops;

p2=p1->next\_shop;

q1=p2->goodslist;

q2=q1->next\_goods;

while(p2)

{

if(strcmp(p2->ShopName,BuyShopName)==0)

{

while(q2)

{

if(strcmp(q2->product.Goods\_name,BuyGoodsName)==0)

{

q2->product.Sales+=count;

printf("%s %d\n",q2->product.Goods\_name,q2->product.Sales);

break;

}

q1=q2;

q2=q1->next\_goods;

}

if(q2==NULL)

{

printf("该商铺中不存在该商品\n");

}

break;

}

p1=p2;

p2=p1->next\_shop;

if(p2!=NULL)

{

q1=p2->goodslist;

q2=q1->next\_goods;

}

}

if(p2==NULL)

{

printf("不存在该商铺\n");

}

return Shops;

}

void InitSaleList(SaleList \*sale)

{

sale->base=(SaleNode \*)malloc(LIST\_INIT\_SIZE\*sizeof(SaleNode));

if(!sale->base)

{

printf("创建顺序表失败！");

exit(0);

}

sale->length=0;

sale->listsize=LIST\_INIT\_SIZE;

}

void InsertSaleList(SaleList \*sale,GoodsNode \*flag)

{

SaleNode \*newbase;

if(sale->length>=sale->listsize)

{

newbase=(SaleNode \*)realloc(sale->base,(sale->listsize+LISTINCERMENT)\*sizeof(SaleNode));

sale->base=newbase;

sale->listsize+=LISTINCERMENT;

}

if((sale->base+sale->length)->GoodsName);

if(flag->product.Goods\_name);

strcpy(sale->base[sale->length].GoodsName,flag->product.Goods\_name);

(sale->base+sale->length)->Sales=flag->product.Sales;

sale->length++;

}

void SaleChange(SaleList \*sale,GoodsNode \*flag)

{

int i;

for(i=0;i<sale->length;i++)

{

if(strcmp((sale->base+i)->GoodsName,flag->product.Goods\_name)==0)

{

(sale->base+i)->Sales+=flag->product.Sales;

break;

}

}

}

GoodsNode\* ExistInSaleList(SaleList sale,GoodsNode \*flag)

{

int i;

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

{

if(strcmp((sale.base+i)->GoodsName,flag->product.Goods\_name)==0)

{

return flag;

}

}

return NULL;

}

void OutputSaleList(SaleList sale)

{

int i;

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

{

printf("%d:%s %d\n",i+1,(sale.base+i)->GoodsName,(sale.base+i)->Sales);

}

}

void SaleSort(ShopList Shops)

{

SaleList sale;

InitSaleList(&sale);

Shop \*p1,\*p2;

p1=Shops;

p2=p1->next\_shop;

GoodsNode \*q1,\*q2,\*flag;

q1=p2->goodslist;

q2=q1->next\_goods;

while(p2)

{

while(q2)

{

flag=ExistInSaleList(sale,q2);

if(flag)

{

SaleChange(&sale,q2);

}

else InsertSaleList(&sale,q2);

q1=q2;

q2=q1->next\_goods;

}

p1=p2;

p2=p1->next\_shop;

if(p2!=NULL)

{

q1=p2->goodslist;

q2=q1->next\_goods;

}

}

OutputSaleList(sale);

}

void PrintShops(ShopList Shops)

{

FILE \*output;

output=fopen("ShopsOut.txt","w");

if(output==NULL)

{

printf("Can't open the file ShopsIn.txt\n");

exit(0);

}

Shop \*p1,\*p2;

GoodsNode \*q1,\*q2;

p1=Shops;

p2=p1->next\_shop;

q1=p2->goodslist;

q2=q1->next\_goods;

fprintf(output,"商铺编号，商铺名，商铺信誉，括号内为商品和商品价格以及其销量\n");

while(p2)

{

fprintf(output,"%d %s %d",p2->ShopNum,p2->ShopName,p2->reputation);

while(q2)

{

fprintf(output,"(%s %.2f %d) ",q2->product.Goods\_name,q2->product.Price,q2->product.Sales);

q1=q2;

q2=q1->next\_goods;

}

fprintf(output,"\n");

p1=p2;

p2=p1->next\_shop;

if(p2!=NULL)

{

q1=p2->goodslist;

q2=q1->next\_goods;

}

}

fclose(output);

}

int main()

{

ShopList Shops;

int m=1;

while(m!=0){

system("cls");

system("date/t");

system("time/t");

printf("学号:031720106\n姓名:徐宇顺");

printf("请输入需要进行的操作：\n");

printf("1.建造商铺链\n");

printf("2.增加商铺\n");

printf("3.删除商铺\n");

printf("4.在所有商铺中查询某个商品\n");

printf("5.购买商品\n");

printf("6.输出所有的商品和销量\n");

printf("0.exit\n");

scanf("%d",&m);

switch(m)

{

case 1:{

system("cls");

Shops=InitShop(Shops);

PrintShops(Shops);

system("pause");

break;

}

case 2:{

system("cls");

Shops=InsertShop(Shops);

PrintShops(Shops);

system("pause");

break;

}

case 3:{

system("cls");

Shops=DeleteShop(Shops);

PrintShops(Shops);

system("pause");

break;

}

case 4:{

system("cls");

SearchGood(Shops);

system("pause");

break;

}

case 5:{

system("cls");

Shops=BuyGoods(Shops);

PrintShops(Shops);

system("pause");

break;

}

case 6:{

system("cls");

SaleSort(Shops);

system("pause");

break;

}

case 0:{

printf("已成功退出！\n");

exit(0);

break;

}

}

}

return 0;

}

# 二、公共钥匙盒

2.1 题目简介

有一个学校的老师共用N个教室，按照规定，所有的钥匙都必须放在公共钥匙盒里，老师不能带钥匙回家。每次老师上课前，都从公共钥匙盒里找到自己上课的教室的钥匙去开门，上完课后，再将钥匙放回到钥匙盒中。

　　钥匙盒一共有N个挂钩，从左到右排成一排，用来挂N个教室的钥匙。一串钥匙没有固定的悬挂位置，但钥匙上有标识，所以老师们不会弄混钥匙。

　　每次取钥匙的时候，老师们都会找到自己所需要的钥匙将其取走，而不会移动其他钥匙。每次还钥匙的时候，还钥匙的老师会找到最左边的空的挂钩，将钥匙挂在这个挂钩上。如果有多位老师还钥匙，则他们按钥匙编号从小到大的顺序还。如果同一时刻既有老师还钥匙又有老师取钥匙，则老师们会先将钥匙全还回去 再取出。

　　今天开始的时候钥匙是按编号从小到大的顺序放在钥匙盒里的。有K位老师要上课，给出每位老师所需要的钥匙、开始上课的时间和上课的时长，假设下课时间就是还钥匙时间，请问最终钥匙盒里面钥匙的顺序是怎样的？

[基本要求]

输入格式

　　输入的第一行包含两个整数N, K。

　　接下来K行，每行三个整数w, s, c，分别表示一位老师要使用的钥匙编号、开始上课的时间和上课的时长。可能有多位老师使用同一把钥匙，但是老师使用钥匙的时间不会重叠。

　　保证输入数据满足输入格式，你不用检查数据合法性。

输出格式

　　输出一行，包含N个整数，相邻整数间用一个空格分隔，依次表示每个挂钩上挂的钥匙编号。

样例输入

5 2

4 3 3

2 2 7

样例输出

1 4 3 2 5

样例说明

　　第一位老师从时刻3开始使用4号教室的钥匙，使用3单位时间，所以在时刻6还钥匙。第二位老师从时刻2开始使用钥匙，使用7单位时间，所以在时刻9还钥匙。

　　每个关键时刻后的钥匙状态如下（X表示空）：

　　时刻2后为1X345；

　　时刻3后为1X3X5；

　　时刻6后为143X5；

　　时刻9后为14325。

课程设计要求：

（1）要求从文本文件中输入；

（2）根据时间进程，将取走钥匙和归还钥匙分别视为事件，放入队列中，然后通过每个事件的先后发生对钥匙盒的状态进行变更；

（3）严格按照要求的输入输出格式进行数据的输入、输出（训练CSP考试中的格式化输入输出的正确性）；

（4）选做：通过图形界面来显示钥匙盒的即时状态，以及事件队列的状态。

2.2 数据结构

typedef struct

{

int \*key;

int length;

}Keys;

Keys中所存储的是所有的钥匙

typedef struct

{

int key\_num;

int start\_time;

int end\_time;

}Classes;

Classes中存储的是一个课堂所要使用的钥匙，以及一个课堂开始时间，和课堂结束的时间

typedef struct

{

Classes \*base;

int length;

}Class\_List;

用一个顺序结构存储所有的课堂

typedef struct QNode

{

Classes data;

struct QNode \*next;

}QNode,\*QueuePtr;

建立队列

typedef struct Link

{

QueuePtr front;

QueuePtr rear;

}LinkQueue;

//队列

2.3 算法设计思想

建立一个以每个课堂开始时间从小到大排序的队列，和一个以每个课堂结束时间从小到大的队列。

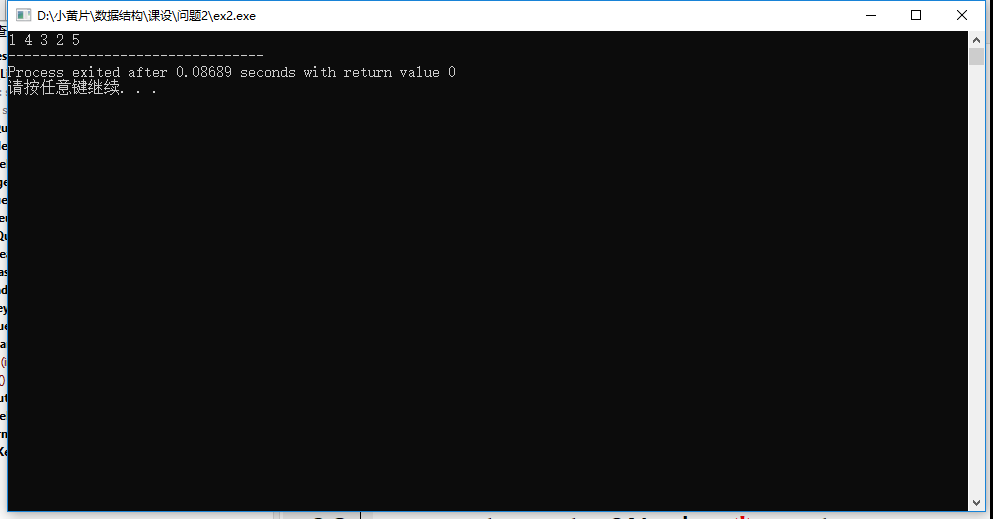
记录一个time来作为时间的变量

以课堂开始时间的队列开始，首先出队，比较课堂结束时间的队头，如果课堂结束时间的队头小于或者是等于课堂开始时间的队头的时间，则课堂结束时间的队列出列，且进行还钥匙的操作。若课堂开始时间小于课堂结束时间则进行取钥匙的操作。

直到最后两个队列全部出队。

取钥匙和还钥匙的操作全部进行完成。

2.4测试数据和结果

2.5算法时间复杂度

InitKeys(Keys \*keys,int N)

时间复杂度O(n)

InitStartQueue(LinkQueue \*Q,Class\_List classlist)

时间复杂度O(n)

InitEndQueue(LinkQueue \*Q,Class\_List classlist)

时间复杂度O(n)

ReturnKeys(Keys \*keys,Classes end)

时间复杂度O(n)

TakeKeys(Keys \*keys,Classes start)

时间复杂度O(n)

ChangeKeys(Class\_List \*classlist,Keys \*keys)

时间复杂度O(n)

2.6源代码

**#include<stdio.h>**

**#include<malloc.h>**

**#include<stdlib.h>**

**#include<string.h>**

**typedef struct**

**{**

**int \*key;**

**int length;**

**}Keys;**

**typedef struct**

**{**

**int key\_num;**

**int start\_time;**

**int end\_time;**

**}Classes;**

**typedef struct**

**{**

**Classes \*base;**

**int length;**

**}Class\_List;**

**typedef struct QNode**

**{**

**Classes data;**

**struct QNode \*next;**

**}QNode,\*QueuePtr;**

**typedef struct Link**

**{**

**QueuePtr front;**

**QueuePtr rear;**

**}LinkQueue;**

**void InitKeys(Keys \*keys,int N)**

**{**

**keys->key=(int \*)malloc(N\*sizeof(int));**

**if(!keys->key)**

**{**

**printf("创建失败\n");**

**exit(0);**

**}**

**keys->length=N;**

**int i;**

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

**{**

**\*(keys->key+i)=i+1;**

**}**

**}**

**void InitClassList(Class\_List \*\*classlist,int K)**

**{**

**(\*classlist)->base=(Classes \*)malloc(K\*sizeof(Classes));**

**if(!(\*classlist)->base)**

**{**

**printf("创建失败\n");**

**exit(0);**

**}**

**(\*classlist)->length=K;**

**}**

**Keys Input(int \*N,int \*K,Class\_List \*classlist)**

**{**

**FILE \*in;**

**int i;**

**in=fopen("Input.txt","r");**

**if(in==NULL)**

**{**

**printf("Can't open the file Input.txt\n");**

**exit(0);**

**}//从文件中读入**

**fscanf(in,"%d%d",N,K);**

**Keys keys;**

**int during\_time;**

**InitKeys(&keys,\*N);**

**InitClassList(&classlist,\*K);**

**for(i=0;i<\*K;i++)**

**{**

**fscanf(in,"%d%d%d",&classlist->base[i].key\_num,&classlist->base[i].start\_time,&during\_time);**

**classlist->base[i].end\_time=classlist->base[i].start\_time+during\_time;**

**}**

**fclose(in);**

**return keys;**

**}**

**void InitQueue(LinkQueue \*Q)**

**{**

**Q->front=(QueuePtr)malloc(sizeof(QNode));**

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

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

**}//创建队列**

**void EnQueue(LinkQueue \*Q,Classes e)**

**{**

**QNode \*p;**

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

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

**p->data=e;**

**Q->rear=p;**

**p->next=NULL;**

**}**

**void DeQueue(LinkQueue \*Q,Classes \*e)**

**{**

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

**{**

**printf("该队列为空");**

**return ;**

**}**

**QNode \*p,\*q;**

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

**\*e=p->data;**

**q=p->next;**

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

**if(Q->rear==p) Q->rear=Q->front;**

**free(p);**

**}**

**int ExsitQueue(LinkQueue \*Q,Classes flag)**

**{**

**QNode \*p,\*q;**

**p=Q->front;**

**q=p->next;**

**while(q)**

**{**

**if(flag.key\_num=q->data.key\_num&&flag.start\_time==q->data.start\_time&&flag.end\_time==q->data.end\_time)**

**return 1;**

**p=q;**

**q=p->next;**

**}**

**return 0;**

**}**

**void InitStartQueue(LinkQueue \*Q,Class\_List classlist)**

**{**

**int i,j;**

**int min\_StartTime;**

**int max\_StartTime=0;**

**Classes flag;**

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

**{**

**if(classlist.base[i].start\_time>max\_StartTime)**

**{**

**max\_StartTime=classlist.base[i].start\_time;**

**}**

**}**

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

**{**

**min\_StartTime=max\_StartTime;**

**for(j=0;j<classlist.length;j++)**

**{**

**if(!ExsitQueue(Q,classlist.base[j])&&classlist.base[j].start\_time<=min\_StartTime)**

**{**

**flag=classlist.base[j];**

**min\_StartTime=classlist.base[j].start\_time;**

**}**

**}**

**EnQueue(Q,flag);**

**}**

**}**

**void InitEndQueue(LinkQueue \*Q,Class\_List classlist)**

**{**

**int i,j;**

**int min\_EndTime;**

**int max\_EndTime=0;**

**Classes flag1,flag2;**

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

**{**

**if(classlist.base[i].end\_time>max\_EndTime)**

**{**

**max\_EndTime=classlist.base[i].end\_time;**

**flag2=classlist.base[i];**

**}**

**}**

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

**{**

**min\_EndTime=max\_EndTime;**

**flag1=flag2;**

**for(j=0;j<classlist.length;j++)**

**{**

**if(!ExsitQueue(Q,classlist.base[j])&&classlist.base[j].end\_time<=min\_EndTime)**

**{**

**if(classlist.base[j].end\_time==min\_EndTime&&classlist.base[j].key\_num>flag1.key\_num)**

**{**

**continue;**

**}**

**flag1=classlist.base[j];**

**min\_EndTime=classlist.base[j].end\_time;**

**}**

**}**

**EnQueue(Q,flag1);**

**}**

**}**

**void GetHead(LinkQueue \*Q,Classes \*e)**

**{**

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

**{**

**return;**

**}**

**QNode \*p;**

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

**\*e=p->data;**

**}**

**int QueueEmpty(LinkQueue Q)**

**{**

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

**return 1;**

**}**

**else return 0;**

**}**

**Keys \*ReturnKeys(Keys \*keys,Classes end)**

**{**

**int i=0;**

**while(i<keys->length)**

**{**

**if(keys->key[i]==-1)**

**{**

**keys->key[i]=end.key\_num;**

**break;**

**}**

**i++;**

**}**

**return keys;**

**}**

**Keys \*TakeKeys(Keys \*keys,Classes start)**

**{**

**int i=0;**

**while(i<keys->length)**

**{**

**if(keys->key[i]==start.key\_num)**

**{**

**keys->key[i]=-1;**

**break;**

**}**

**i++;**

**}**

**return keys;**

**}**

**void OutPutKeyList(Keys \*keys)**

**{**

**int i=0;**

**while(i<keys->length)**

**{**

**printf("%d ",keys->key[i]);**

**i++;**

**}**

**}**

**void ChangeKeys(Class\_List \*classlist,Keys \*keys)**

**{**

**LinkQueue StartSortQueue,EndSortQueue;**

**InitQueue(&StartSortQueue);**

**InitQueue(&EndSortQueue);**

**InitStartQueue(&StartSortQueue,\*classlist);**

**InitEndQueue(&EndSortQueue,\*classlist);**

**int time=1;**

**Classes start,end;**

**GetHead(&EndSortQueue,&end);**

**while(!QueueEmpty(StartSortQueue))**

**{**

**DeQueue(&StartSortQueue,&start);**

**time=start.start\_time;**

**while(end.end\_time<=time)**

**{**

**keys=ReturnKeys(keys,end);**

**DeQueue(&EndSortQueue,&end);**

**GetHead(&EndSortQueue,&end);**

**}//先还完所有的钥匙**

**keys=TakeKeys(keys,start);//再取钥匙**

**}**

**while(!QueueEmpty(EndSortQueue))**

**{**

**keys=ReturnKeys(keys,end);**

**DeQueue(&EndSortQueue,&end);**

**GetHead(&EndSortQueue,&end);**

**}**

**OutPutKeyList(keys);**

**}**

**int main()**

**{**

**int N,K;**

**Keys keys;**

**Class\_List classlist;**

**keys=Input(&N,&K,&classlist);**

**ChangeKeys(&classlist,&keys);**

**return 0;**

**}**

# 三、JSON

3.1 题目简介

JSON (JavaScript Object Notation) 是一种轻量级的数据交换格式，可以用来描述半结构化的数据。JSON 格式中的基本单元是值 (value)，出于简化的目的本题只涉及 2 种类型的值：

　　\* 字符串 (string)：字符串是由双引号 " 括起来的一组字符（可以为空）。如果字符串的内容中出现双引号 "，在双引号前面加反斜杠，也就是用 \" 表示；如果出现反斜杠 \，则用两个反斜杠 \\ 表示。反斜杠后面不能出现 " 和 \ 以外的字符。例如：""、"hello"、"\"\\"。

　　\* 对象 (object)：对象是一组键值对的无序集合（可以为空）。键值对表示对象的属性，键是属性名，值是属性的内容。对象以左花括号 { 开始，右花括号 } 结束，键值对之间以逗号 , 分隔。一个键值对的键和值之间以冒号 : 分隔。键必须是字符串，同一个对象所有键值对的键必须两两都不相同；值可以是字符串，也可以是另一个对象。例如：{}、{"foo": "bar"}、{"Mon": "weekday", "Tue": "weekday", "Sun": "weekend"}。

　　除了字符串内部的位置，其他位置都可以插入一个或多个空格使得 JSON 的呈现更加美观，也可以在一些地方换行，不会影响所表示的数据内容。例如，上面举例的最后一个 JSON 数据也可以写成如下形式。

　　{

　　"Mon": "weekday",

　　"Tue": "weekday",

　　"Sun": "weekend"

　　}

　　给出一个 JSON 格式描述的数据，以及若干查询，编程返回这些查询的结果。

输入格式

　　第一行是两个正整数 n 和 m，分别表示 JSON 数据的行数和查询的个数。

　　接下来 n 行，描述一个 JSON 数据，保证输入是一个合法的 JSON 对象。

　　接下来 m 行，每行描述一个查询。给出要查询的属性名，要求返回对应属性的内容。需要支持多层查询，各层的属性名之间用小数点 . 连接。保证查询的格式都是合法的。

[基本要求]

输出格式

　　对于输入的每一个查询，按顺序输出查询结果，每个结果占一行。

　　如果查询结果是一个字符串，则输出 STRING <string>，其中 <string> 是字符串的值，中间用一个空格分隔。

　　如果查询结果是一个对象，则输出 OBJECT，不需要输出对象的内容。

　　如果查询结果不存在，则输出 NOTEXIST。

样例输入

10 5

{

"firstName": "John",

"lastName": "Smith",

"address": {

"streetAddress": "2ndStreet",

"city": "NewYork",

"state": "NY"

},

"esc\\aped": "\"hello\""

}

firstName

address

address.city

address.postal

esc\aped

样例输出

STRING John

OBJECT

STRING NewYork

NOTEXIST

STRING "hello"

[基本要求]

（1）要求从文本文件中输入；

（2）本题目其实就是一棵普通的树（即每个结点的孩子数不固定，不能单纯采用n叉树来解决），可以考虑使用孩子兄弟表示法等进行表示和存储；

（3）严格按照要求的输入输出格式进行数据的输入、输出（训练CSP考试中的格式化输入输出的正确性）；

（4）选做：使用图形界面（或字符格式化摆成的树形结构，参考Linux下的tree命令），以树状形式显示输入的JSON格式数据。

3.2 数据结构

typedef struct bitnode

{

char KeyWords[80];

char type[20];

struct bitnode \*Child;

struct bitnode \*Sibling;

}BiTNode,\*BiTree;

树中每个结点由树的类型，以及对应的字符串

typedef struct SearchWords

{

char string[80];

}SearchWords;

typedef struct

{

SearchWords \*base;

int length;

}SearchList;

查找的顺序表

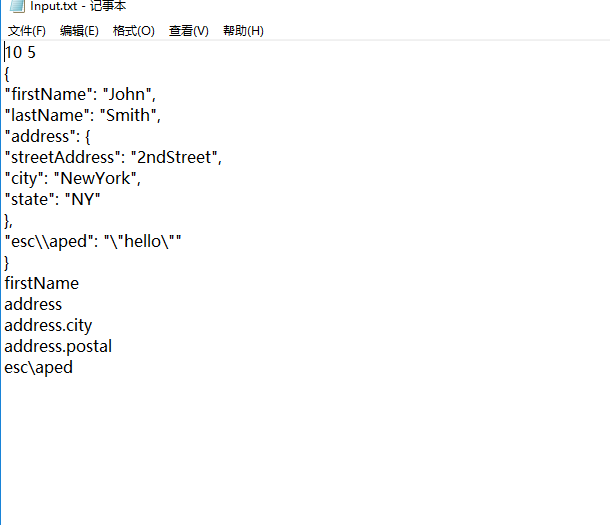
3.3 算法设计思想

在建立树的时候，对象里面的元素彼此之间是Sibling

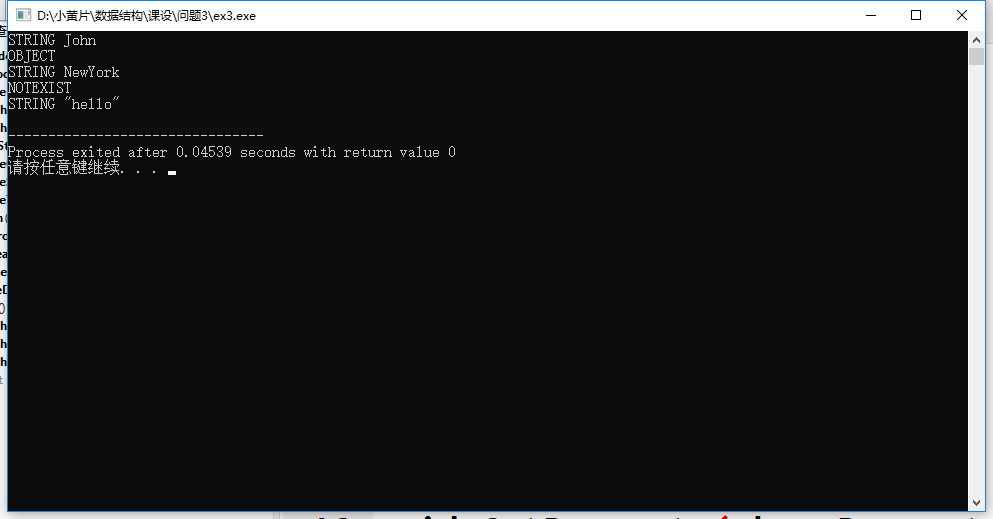
以及每一个元素里面的的Child是对应的键的键值

在查询的时候，要先判断有没有小数点，如果有小数点，则必定是在一个对象里面寻找，如果不存在则输出NOEXIST，如果存在就返回对应的键值。在对应的对象里面查找的时候，要继续判断是否有另外的小数点，如果有的话，则需要递归在对象里面进行查找。如果起初没有小数点，则直接在树中查找是否有对应的键关键字，如果没有的话，则输出NOEXIST，如果存在则输出STRING+对应的键值。

3.4测试数据和结果



此为输入



输出为此

3.5算法时间复杂度

SearchInObject(char search[],char result[],BiTNode \*p)

时间复杂度O(n2)

3.6源代码

#include<stdio.h>

#include<malloc.h>

#include<stdlib.h>

#include<string.h>

typedef struct bitnode

{

char KeyWords[80];

char type[20];

struct bitnode \*Child;

struct bitnode \*Sibling;

}BiTNode,\*BiTree;

typedef struct SearchWords

{

char string[80];

}SearchWords;

typedef struct

{

SearchWords \*base;

int length;

}SearchList;

int n,m;

BiTree InitTree()

{

BiTree T;

T=(BiTree)malloc(sizeof(BiTNode));

T->Child=NULL;

T->Sibling=NULL;

return T;

}//初始化树

void Getch(char \*ch,FILE \*in)

{

(\*ch)=fgetc(in);

while(\*ch==' '||\*ch==',')

{

\*ch=fgetc(in);

}

}

void GetProperty(char Property[],FILE \*in)

{

int i=0;

char ch,flag=' ';

ch=fgetc(in);

while(ch!='"')

{

if(ch=='\\')

{

flag=ch;

ch=fgetc(in);

if(!(ch=='"'||ch=='\\'))

{

printf("输入有误");

exit(0);

}

Property[i]=ch;

}

Property[i]=ch;

i++;

ch=fgetc(in);

flag=' ';

}

}

void ClearString(char string[])

{

int i=0;

while(string[i]!='\0')

{

string[i]='\0';

i++;

}

}

BiTNode \*CreateString(BiTNode \*p1,FILE \*in)

{

BiTNode \*q;

char value[80];

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

GetProperty(value,in);

q->Child=NULL;

q->Sibling=NULL;

strcpy(q->KeyWords,value);

p1->Child=q;

return p1;

}

int CreateObject(BiTNode \*\*Root,FILE \*in,int i,int n)

{

char ch1,ch2;

char Property[80];//键对中的“键”的部分由property表示

BiTNode \*Childs;

BiTNode \*q1,\*q2;

int cnt;

Getch(&ch1,in);

cnt=i;

while(cnt<=n&&ch1!='}')

{

if(ch1=='\n'){

cnt++;

if(cnt<=n) Getch(&ch1,in);//读取下一行的字符

continue;

}

GetProperty(Property,in);

Getch(&ch2,in);//得到冒号

Getch(&ch1,in);//观察得到的是{还是“

if(ch1=='"')

{

q2=(BiTNode \*)malloc(sizeof(BiTNode));

q2->Child=NULL;

q2->Sibling=NULL;

strcpy(q2->KeyWords,Property);

q2=CreateString(q2,in);

strcpy(q2->type,"STRING");

q2->Sibling=(\*Root)->Child;

(\*Root)->Child=q2;

}

if(ch1=='{'){

q2=(BiTNode \*)malloc(sizeof(BiTNode));

q2->Child=NULL;

q2->Sibling=NULL;

strcpy(q2->KeyWords,Property);

cnt=CreateObject(&q2,in,cnt,n);

strcpy(q2->type,"OBJECT");

q2->Sibling=(\*Root)->Child;

(\*Root)->Child=q2;

}

Getch(&ch1,in);

ClearString(Property);

}

return cnt;

}

BiTree CreateTree(BiTree T,FILE \*in)

{

int i=1;

char ch1,ch2,flag;

Getch(&ch2,in);

Getch(&ch1,in);

while(i<=n)

{

if(ch1=='\n') i++;

if(ch1=='{'){

i=CreateObject(&T,in,i,n);

}

}

return T;

}

void InitSearchList(SearchList \*Slist,FILE \*in)

{

Slist->base=(SearchWords\*)malloc(m\*sizeof(SearchWords));

Slist->length=m;

int i=0,j;

char ch;

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

{

ch=fgetc(in);

j=0;

while(ch!='\n')

{

(Slist->base+i)->string[j]=ch;

ch=fgetc(in);

j++;

}

(Slist->base+i)->string[j]='\0';

}

}

int JudgeDot(char search[],char father[],char son[])

{

int i=0,j;

while(search[i]!='\0')

{

if(search[i]=='.')

{

strncpy(father,search,i);

j=i+1;

while(search[j]!='\0')

{

j++;

}

strncpy(son,&(search[i+1]),j-i-1);

son[j-i-1]='\0';

return 1;

}

i++;

}

return 0;

}

int SearchInTree(char search[],char result[],BiTree T,BiTNode \*\*flag)

{

if(T==NULL)

{

\*flag=NULL;

return 0;

}

if(strcmp(search,T->KeyWords)==0&&strcmp("STRING",T->type)==0)

{

strcpy(result,T->Child->KeyWords);

\*flag=T;

return 1;

}

else

{

if(strcmp(search,T->KeyWords)==0&&strcmp("OBJECT",T->type)==0)

{

\*flag=T;

return 2;

}

else

{

SearchInTree(search,result,T->Sibling,flag);

}

}

}

int SearchInObject(char search[],char result[],BiTNode \*p)

{

int tag=-1;

BiTNode \*q;

char father[80],son[80];

if(!JudgeDot(search,father,son))

{

tag=SearchInTree(search,result,p,&q);

return tag;

}

else{

tag=SearchInTree(father,result,p,&q);

if(tag==0)

{

return tag;

}

tag=SearchInObject(son,result,q->Child);

}

return tag;

}

void Search(SearchList \*Slist,BiTree T)

{

BiTNode \*p;

p=T->Child;

int i,tag=-1;

int judge=0;

char out[80],father[80],son[80];

for(i=0;i<Slist->length;i++)

{

if(!JudgeDot((Slist->base+i)->string,father,son))

{

tag=SearchInTree((Slist->base+i)->string,out,T->Child,&p);

if(tag==1) printf("STRING %s\n",out);

else if(tag==2) printf("OBJECT\n");

else if(tag==0) printf("NOTEXIST\n");

}

else

{

tag=SearchInTree(father,out,T->Child,&p);

if(tag==0) printf("NOTEXIST\n");

else

{

tag=SearchInObject(son,out,p->Child);

if(tag==1) printf("STRING %s\n",out);

else if(tag==2) printf("OBJECT\n");

else if(tag==0) printf("NOTEXIST\n");

}

ClearString(father);

ClearString(son);

}

ClearString(out);

tag=-1;

p=T->Child;

}

}

int main()

{

FILE \*in;

SearchList Slist;

in=fopen("Input.txt","r");

if(in==NULL)

{

printf("Can't open the file Input.txt\n");

exit(0);

}

fscanf(in,"%d%d",&n,&m);

BiTree T;

T=InitTree();

T=CreateTree(T,in);

InitSearchList(&Slist,in);

Search(&Slist,T);

return 0;

}

# 四、Huffman编码和解码

4.1 题目简介

对一篇不少于2000字符的英文文章（source.txt），统计各字符出现的次数，实现Huffman编码(code.dat)，以及对编码结果的解码(recode.txt)。

[基本要求]

（1） 输出每个字符出现的次数和编码,并存储文件(Huffman.txt)。

（2） 在Huffman编码后，英文文章编码结果保存到文件中(code.dat)，编码结果必须是二进制形式，即0 1的信息用比特位表示，不能用字符’0’和’1’表示。

（3） 实现解码功能。

4.2 数据结构

typedef struct {

int \*base;

int length;

int listsize;

}Hnode;

typedef struct{

int frequency;

char str;

Hnode node;

int parent,lchild,rchild;

}HTNode;

typedef struct{

HTNode \*base;

int LeafLength;

int TotalLength;

int Maxsize;

}HuffmanTree;

4.3 算法设计思想

一一读取文件中的字符，如果字符串在存储的顺序表里面存在的话，则将Frequency即频率加一，如果不存在的话，则要在Huffman树中加一个新节点，即一个新的叶子。

叶子输入完毕之后，则开始建立Huffman树，从叶子的下一个空间开始，找寻这个空间之前Frequency最小的两个，小一点的值放在右边，该两个值的Parent都是该空间。

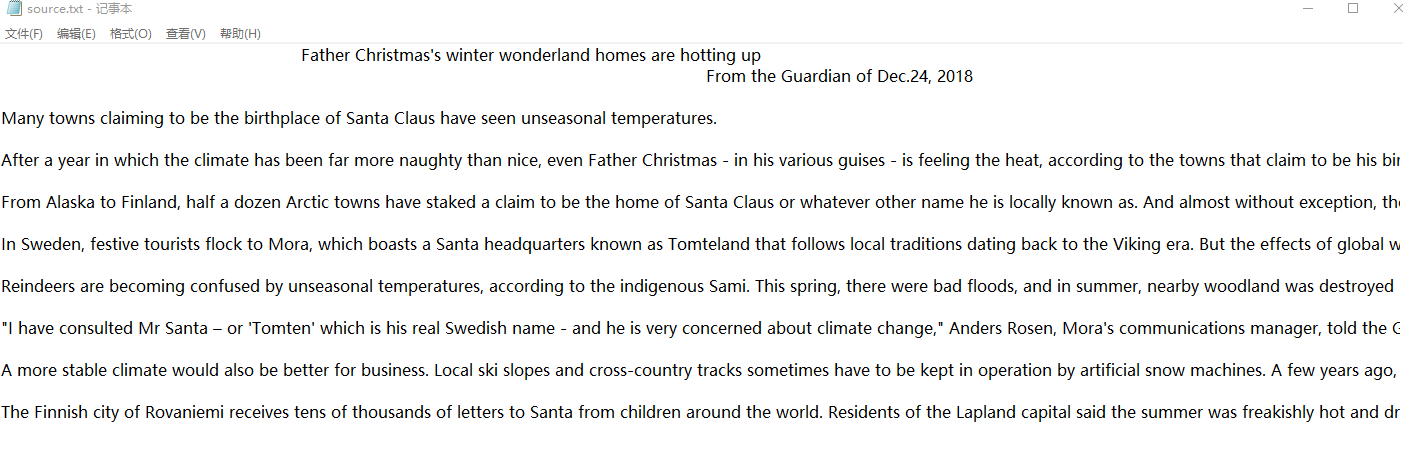
知道最后空间总数为叶子的总数\*2-1。

编码的过程是，左子树添加一个0，右子树添加一个1，进行编码的时候，首先要将Parent里面的编码传递给子类，再在左子树添加一个0，再在右子树添加一个1。

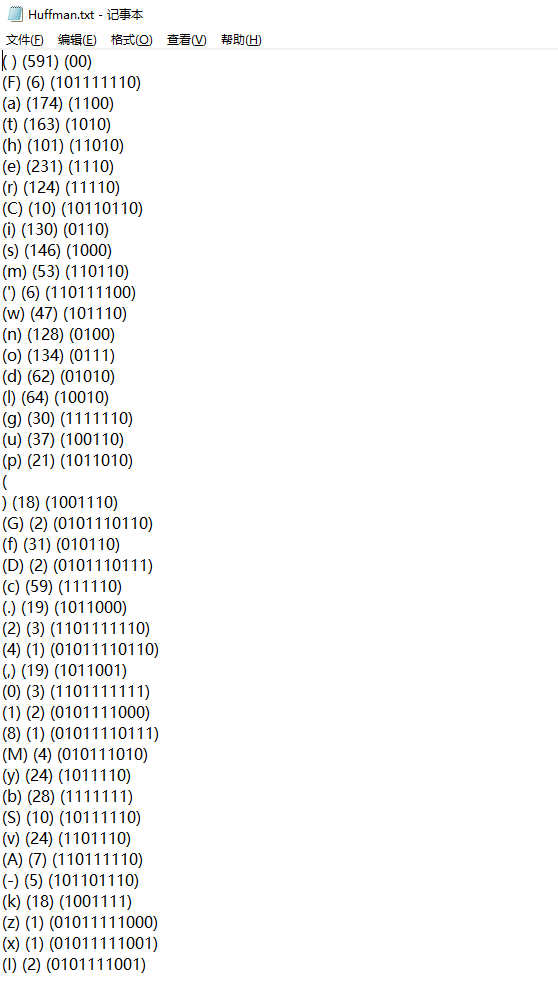
最后一遍读取源文件，读取源文件的同时，在存储列表里面找到对应的字符，在对应的编码到code.dat中。

在重解码的时候，读取code.dat中的编码，将其存储到一个字符型的数组中，每读一个则比较存储结构的每个编码，如果存在的话，则输出到对应的recode.txt中，并且清空所有的字符串数组供下一次读取，若无的话，则读取文件中的下一个编码。

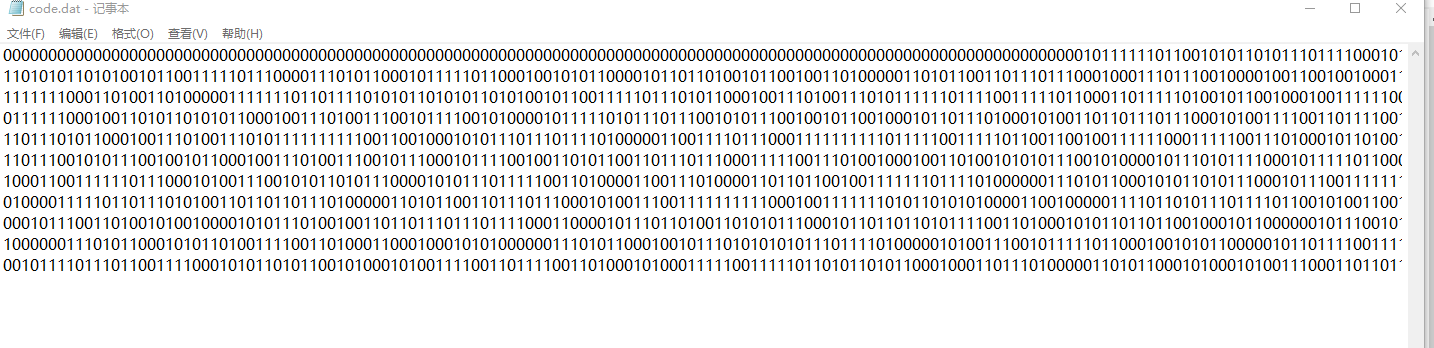
4.4测试数据和结果



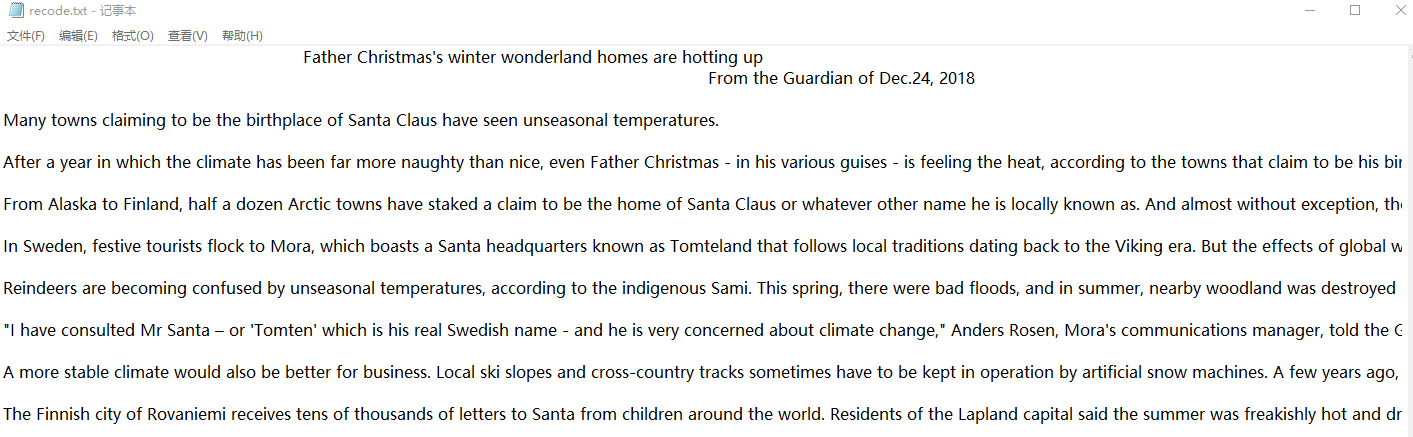
此为源文件



Huffman.txt中存储对应的字符和编码



此为编码



此为重解码

4.5算法时间复杂度

CreateHuffmanTree(HuffmanTree \*HT,FILE \*Input)

时间复杂度O(n);

CreateCode(HuffmanTree \*HT,int point)

时间复杂度O(n2)

4.6源代码

**#include<stdio.h>**

**#include<malloc.h>**

**#include<stdlib.h>**

**#include<string.h>**

**#define LIST\_INIT\_SIZE 100**

**#define LISTINCERMENT 10**

**typedef struct {**

**int \*base;**

**int length;**

**int listsize;**

**}Hnode;**

**typedef struct{**

**int frequency;**

**char str;**

**Hnode node;**

**int parent,lchild,rchild;**

**}HTNode;**

**typedef struct{**

**HTNode \*base;**

**int LeafLength;**

**int TotalLength;**

**int Maxsize;**

**}HuffmanTree;**

**void ExsitInHuffmanTree(HuffmanTree \*\*HT,char ch,int \*flag)**

**{**

**int i;**

**for(i=0;i<(\*HT)->LeafLength;i++)**

**{**

**if(ch==((\*HT)->base+i)->str)**

**{**

**((\*HT)->base+i)->frequency++;**

**\*flag=1;**

**return;**

**}**

**}**

**\*flag=0;**

**}**

**void InitHuff(HuffmanTree \*\*HT)**

**{**

**(\*HT)->base=(HTNode \*)malloc(LIST\_INIT\_SIZE\*sizeof(HTNode));**

**(\*HT)->LeafLength=0;**

**(\*HT)->TotalLength=0;**

**(\*HT)->Maxsize=LIST\_INIT\_SIZE;**

**}**

**void InsertHuffmanTree(HuffmanTree \*\*HT,char leaf)**

**{**

**HTNode \*newbase;**

**HTNode \*p;**

**if((\*HT)->LeafLength>=(\*HT)->Maxsize)**

**{**

**newbase=(HTNode \*)realloc((\*HT)->base,((\*HT)->Maxsize+LISTINCERMENT)\*sizeof(HTNode));**

**(\*HT)->base=newbase;**

**(\*HT)->Maxsize+=LISTINCERMENT;**

**}**

**p=((\*HT)->base+(\*HT)->LeafLength);**

**p->lchild=-1;**

**p->rchild=-1;**

**p->parent=-1;**

**p->str=leaf;**

**p->frequency=1;**

**p->node.base=(int \*)malloc(LIST\_INIT\_SIZE\*sizeof(int));**

**p->node.length=0;**

**p->node.listsize=LIST\_INIT\_SIZE;**

**(\*HT)->LeafLength++;**

**}**

**void Select(HuffmanTree HT,int \*s1,int \*s2,int i)**

**{**

**int j,min1,min2;**

**HTNode \*p;**

**p=HT.base;**

**min1=2000;**

**min2=2000;**

**for(j=0;j<i;p++,j++)**

**{**

**if(p->parent==-1&&min1>p->frequency)**

**{**

**min1=p->frequency;**

**\*s1=j;**

**}**

**}**

**p=HT.base;**

**for(j=0;j<i;p++,j++)**

**{**

**if(p->parent==-1&&min2>p->frequency&&j!=\*s1)**

**{**

**min2=p->frequency;**

**\*s2=j;**

**}**

**}**

**}**

**void ExtendSpace(HuffmanTree \*\*HT)**

**{**

**HTNode \*newbase;**

**if((2\*(\*HT)->LeafLength-1)>=(\*HT)->Maxsize)**

**{**

**newbase=(HTNode \*)realloc((\*HT)->base,(2\*(\*HT)->LeafLength-1)\*sizeof(HTNode));**

**(\*HT)->base=newbase;**

**(\*HT)->Maxsize=(2\*(\*HT)->LeafLength-1);**

**}**

**int i=(\*HT)->LeafLength;**

**(\*HT)->TotalLength=(2\*(\*HT)->LeafLength-1);**

**HTNode \*p;**

**p=(\*HT)->base;**

**while(i<(\*HT)->TotalLength)**

**{**

**(p+i)->parent=-1;**

**(p+i)->lchild=-1;**

**(p+i)->rchild=-1;**

**(p+i)->node.base=(int \*)malloc(LIST\_INIT\_SIZE\*sizeof(int));**

**(p+i)->node.length=0;**

**(p+i)->node.listsize=LIST\_INIT\_SIZE;**

**i++;**

**}**

**}**

**void CreateHuffmanTree(HuffmanTree \*HT,FILE \*Input)**

**{**

**char ch;**

**int t;**

**int tag=0,i,m;**

**int s1,s2;**

**InitHuff(&HT);**

**HTNode \*p;**

**ch=fgetc(Input);**

**while(!feof(Input))**

**{**

**ExsitInHuffmanTree(&HT,ch,&tag);//如果存在，就frequency++**

**if(tag==0)**

**{**

**InsertHuffmanTree(&HT,ch);//不存在就插入一个新的叶子**

**}**

**ch=fgetc(Input);**

**}//输入所有的字符**

**ExtendSpace(&HT);//扩大Huffman的空间**

**p=HT->base;**

**m=2\*(HT->LeafLength)-1;**

**for(i=HT->LeafLength;i<m;i++)**

**{**

**Select(\*HT,&s1,&s2,i);//挑选出最小的两个值**

**if(s1>s2)**

**{**

**t=s1;**

**s1=s2;**

**s2=t;**

**}**

**(p+s1)->parent=i;**

**(p+s2)->parent=i;**

**(p+i)->lchild=s1;//小的那个值赋给左子树**

**(p+i)->rchild=s2;**

**(p+i)->frequency=(p+s1)->frequency+(p+s2)->frequency;//构建huffman树**

**}**

**}**

**HTNode \*InsertCode(HTNode \*p,int e)**

**{**

**int m;**

**m=p->node.length;**

**int \*newbase;**

**if(p->node.length>=p->node.listsize)**

**{**

**newbase=(int \*)realloc(p->node.base,(p->node.listsize+LISTINCERMENT)\*sizeof(int));**

**p->node.base=newbase;**

**p->node.listsize+=LISTINCERMENT;**

**}**

**\*((p->node.base)+m)=e;**

**(p->node.length)++;**

**return p;**

**}**

**HTNode \*CopyNode(HTNode \*father,HTNode \*son)**

**{**

**int \*f,\*s;**

**f=father->node.base;**

**s=son->node.base;**

**int i=0;**

**while(i<father->node.length)**

**{**

**\*(s+i)=\*(f+i);**

**son->node.length++;**

**i++;**

**}**

**return son;**

**}**

**void CreateCode(HuffmanTree \*HT,int point)**

**{**

**HTNode \*p,\*q;**

**p=(HT->base+point);**

**if(p->lchild==-1&&p->rchild==-1)//若是叶子了就退出**

**{**

**return;**

**}**

**else{**

**q=(HT->base+p->lchild);**

**q=CopyNode(p,q);//将其父类的编码给到子类**

**q=InsertCode(HT->base+p->lchild,48);//左子树放到0**

**CreateCode(HT,p->lchild);**

**q=(HT->base+p->rchild);**

**q=CopyNode(p,q);**

**q=InsertCode(HT->base+p->rchild,49);//右子树放到1**

**CreateCode(HT,p->rchild);**

**}**

**}**

**void OutputCode(HuffmanTree HT)**

**{**

**HTNode \*p;**

**p=HT.base;**

**FILE \*output;**

**output=fopen("Huffman.txt","w");**

**if(output==NULL)**

**{**

**printf("Can't open the file Huffman.txt\n");**

**exit(0);**

**}**

**int i=0,j;**

**while(i<HT.LeafLength)**

**{**

**j=0;**

**fprintf(output,"(%c) (%d) ",p->str,p->frequency);**

**fprintf(output,"(");**

**while(j<p->node.length)**

**{**

**if(\*(p->node.base+j)==48)**

**{**

**fprintf(output,"0");**

**}**

**else fprintf(output,"1");**

**j++;**

**}**

**fprintf(output,")\n");**

**i++;**

**p=(HT.base+i);**

**}**

**}**

**HTNode \*FindCode(HuffmanTree HT,char ch)**

**{**

**HTNode \*q;**

**q=HT.base;**

**int i=0;**

**while(i<HT.LeafLength)**

**{**

**if(q->str==ch)**

**{**

**return q;**

**}**

**q++;**

**i++;**

**}**

**}**

**void Writecode(FILE \*write,HTNode \*p)**

**{**

**int i=0;**

**int \*q;**

**q=p->node.base;**

**while(i<p->node.length)**

**{**

**fwrite(q,1,1,write);**

**q++;**

**i++;**

**}**

**}**

**void WriteInCode(HuffmanTree HT)**

**{**

**FILE \*readsource;**

**FILE \*write;**

**HTNode \*p;**

**readsource=fopen("source.txt","r");**

**write=fopen("code.dat","wb");**

**if(readsource==NULL)**

**{**

**printf("Can't open the file source.txt\n");**

**exit(0);**

**}**

**if(write==NULL)**

**{**

**printf("Can't open the file code.dat\n");**

**exit(0);**

**}**

**char ch;**

**ch=fgetc(readsource);**

**while(!feof(readsource))**

**{**

**p=FindCode(HT,ch);**

**Writecode(write,p);**

**ch=fgetc(readsource);**

**}**

**fclose(readsource);**

**fclose(write);**

**}**

**int ExsistCode(HTNode \*q,int code[])**

**{**

**int i=0;**

**while(i<q->node.length)**

**{**

**if(code[i]!=\*(q->node.base+i))**

**{**

**return 0;**

**}**

**i++;**

**}**

**return 1;**

**}**

**HTNode \*JudgeCode(HuffmanTree HT,int code[])**

**{**

**HTNode \*q;**

**q=HT.base;**

**int i=0;**

**while(i<HT.LeafLength)**

**{**

**if(ExsistCode(q,code))**

**{**

**return q;**

**}**

**i++;**

**q++;**

**}**

**return NULL;**

**}**

**void ClearCode(int code[])**

**{**

**int i=0;**

**while(code[i]!=0)**

**{**

**code[i]=0;**

**i++;**

**}**

**}**

**void AddCode(FILE \*read,int code[])**

**{**

**int i=0;**

**while(code[i]!=0)**

**{**

**i++;**

**}**

**fread(&code[i],1,1,read);**

**}**

**void ReCode(HuffmanTree HT)**

**{**

**FILE \*recode;**

**FILE \*read;**

**HTNode \*p;**

**p=HT.base;**

**int code[2000]={0};**

**read=fopen("code.dat","rb");**

**recode=fopen("recode.txt","w");**

**if(read==NULL)**

**{**

**printf("Can't open the file code.dat\n");**

**exit(0);**

**}**

**if(recode==NULL)**

**{**

**printf("Can't open the file recode.txt\n");**

**exit(0);**

**}**

**while(!feof(read))**

**{**

**p=JudgeCode(HT,code);**

**if(p!=NULL)**

**{**

**fprintf(recode,"%c",p->str);**

**ClearCode(code);**

**}**

**else**

**{**

**AddCode(read,code);**

**}**

**p=HT.base;**

**}**

**}**

**int main()**

**{**

**HuffmanTree HT;**

**FILE \*Input;**

**Input=fopen("source.txt","r");**

**if(Input==NULL)**

**{**

**printf("Can't open the file source.txt\n");**

**exit(0);**

**}**

**CreateHuffmanTree(&HT,Input);**

**CreateCode(&HT,HT.TotalLength-1);**

**OutputCode(HT);**

**fclose(Input);**

**WriteInCode(HT);**

**ReCode(HT);**

**return 0;**

**}**

# 五、行车路线

5.1 题目简介

小明和小芳出去乡村玩，小明负责开车，小芳来导航。

　　小芳将可能的道路分为大道和小道。大道比较好走，每走1公里小明会增加1的疲劳度。小道不好走，如果连续走小道，小明的疲劳值会快速增加，连续走s公里小明会增加s2的疲劳度。

　　例如：有5个路口，1号路口到2号路口为小道，2号路口到3号路口为小道，3号路口到4号路口为大道，4号路口到5号路口为小道，相邻路口之间的距离都是2公里。如果小明从1号路口到5号路口，则总疲劳值为(2+2)2+2+22=16+2+4=22。

现在小芳拿到了地图，请帮助她规划一个开车的路线，使得按这个路线开车小明的疲劳度最小。

[基本要求]

输入格式：

　　输入的第一行包含两个整数n, m，分别表示路口的数量和道路的数量。路口由1至n编号，小明需要开车从1号路口到n号路口。

接下来m行描述道路，每行包含四个整数t, a, b, c，表示一条类型为t，连接a与b两个路口，长度为c公里的双向道路。其中t为0表示大道，t为1表示小道。保证1号路口和n号路口是连通的。

输出格式

输出一个整数，表示最优路线下小明的疲劳度。

样例输入

6 7

1 1 2 3

1 2 3 2

0 1 3 30

0 3 4 20

0 4 5 30

1 3 5 6

1 5 6 1

样例输出

76

样例说明

　　从1走小道到2，再走小道到3，疲劳度为52=25；然后从3走大道经过4到达5，疲劳度为20+30=50；最后从5走小道到6，疲劳度为1。总共为76。

课程设计要求：

（1）要求从文本文件中输入；

（2）采用适当的数据结构存储由输入数据中的道路所形成的图结构；

（3）编写尽可能优的算法，处理好连续走小道造成的疲劳值的指数增长（提示：基于迪杰斯特拉算法进行改进即可完成本题）；

（4）除严格按题目要求进行输出以外，还要求输出最优路线的路径，以及从出发点到各个点的最小疲劳值。

5.2 数据结构

typedef struct Point

{

int RoadKind;

int point;

int RoadLength;

struct Point \*next;

}Point;

typedef struct Node

{

int node;

Point \*relate;

}Node;

typedef struct PList

{

Node \*base;

int length;

}PList;

typedef struct

{

int point;

int exhaust;

}RoadNode;

typedef struct Road

{

RoadNode \*base;

int length;

}Road;

typedef struct Stack

{

int \*base;

int \*top;

int stacksize;

}Stack;

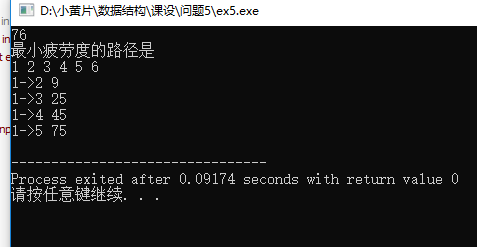
5.3 算法设计思想

在寻找最小的疲劳度时，对于每一个节点找与其相连的点，且到每个点的疲劳度

最后选择一个最小的疲劳度的点，如果这个点找到的是小道，则入栈，如果最后找的是大道，就清空栈。

把最小疲劳度的所有点存储在一个顺序结构里面，最后输出。

5.4测试数据和结果



5.5算法时间复杂度

FindRoad(PList list,int start,int end,int \*totalexhaust,Stack stack,Road \*road)

时间复杂度O(n2)

Froad(PList list,int begin,int \*key,int exhaust,Stack \*stack,Road \*\*road)

时间复杂度O(n2)

5.6源代码

#include<stdio.h>

#include<malloc.h>

#include<stdlib.h>

#define STACK\_INIT\_SIZE 100

#define STACKINCREMENT 10

int n,m;

typedef struct Point

{

int RoadKind;

int point;

int RoadLength;

struct Point \*next;

}Point;

typedef struct Node

{

int node;

Point \*relate;

}Node;

typedef struct PList

{

Node \*base;

int length;

}PList;

typedef struct

{

int point;

int exhaust;

}RoadNode;

typedef struct Road

{

RoadNode \*base;

int length;

}Road;

typedef struct Stack

{

int \*base;

int \*top;

int stacksize;

}Stack;

void InitList(PList \*list)

{

list->base=(Node \*)malloc(n\*sizeof(Node));

if(!list->base)

{

printf("创建点的顺序表失败！\n");

exit(0);

}

list->length=n;

int i=0;

while(i<list->length)

{

(list->base+i)->node=i+1;

(list->base+i)->relate=(Point\*)malloc(sizeof(Point));

(list->base+i)->relate->next=NULL;

i++;

}

}

void InputRelatePoint(PList \*list,FILE \*input)

{

int type,begin,end,roadlength;

fscanf(input,"%d%d%d%d",&type,&begin,&end,&roadlength);

Point \*p;

Node \*q;

q=list->base;

int i=0;

while(!feof(input))

{

while(i<list->length)

{

if(q->node==begin)

{

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

p->RoadKind=type;

p->RoadLength=roadlength;

p->point=end;

p->next=q->relate->next;

q->relate->next=p;//用头插法

}

if(q->node==end)

{

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

p->RoadKind=type;

p->RoadLength=roadlength;

p->point=begin;

p->next=q->relate->next;

q->relate->next=p;

}

i++;

q++;

}

i=0;

q=list->base;

fscanf(input,"%d%d%d%d",&type,&begin,&end,&roadlength);

}

}

void InitRoad(Road \*road)

{

road->base=(RoadNode\*)malloc(n\*(sizeof(RoadNode)));

if(!road->base)

{

printf("创建失败\n");

exit(0);

}

road->length=0;

}

void InitStack(Stack \*stack)

{

stack->base=(int\*)malloc(STACK\_INIT\_SIZE\*sizeof(int));

if(!stack->base)

{

printf("创建栈失败！");

exit(0);

}

stack->top=stack->base;

stack->stacksize=STACK\_INIT\_SIZE;

}

void ClearStack(Stack \*stack)

{

stack->top=stack->base;

}

int TraverseStack(Stack stack)

{

int \*p;

int total=0;

p=stack.base;

while(p!=stack.top)

{

total+=(\*p);

p++;

}

return total;

}

void Push(Stack \*\*stack,int e)

{

if((\*stack)->top-(\*stack)->base==STACK\_INIT\_SIZE)

{

(\*stack)->base=(int \*)realloc((\*stack)->base,((\*stack)->stacksize+STACKINCREMENT)\*sizeof(int));

(\*stack)->top=(\*stack)->base+(\*stack)->stacksize;

(\*stack)->stacksize+=STACKINCREMENT;

}

if(!(\*stack)->base){

printf("插入元素时栈满，重新分配空间出错！");

return;

}

\*((\*stack)->top)=e;

(\*stack)->top++;

}

int ExsitInRoad(Road \*road,int point)

{

RoadNode \*p;

p=road->base;

int i=0;

while(i<road->length)

{

if(p->point==point)

{

return 1;

}

i++;

p++;

}

return 0;

}

int Froad(PList list,int begin,int \*key,int exhaust,Stack \*stack,Road \*\*road)

{

Node \*p;

Point \*q;

p=list.base;

q=p->relate->next;

\*key=0;

int i=0,flag,min=0x100;

int insertlength;

int tag;

int lastexhaust;

while(i<list.length)

{

if(p->node==begin)

{

while(q)

{

if(ExsitInRoad(\*road,q->point))

{

q=q->next;

continue;

}

if(q->RoadKind==0)

{

flag=exhaust+q->RoadLength;

}

if(q->RoadKind==1)

{

lastexhaust=TraverseStack(\*stack);//观察连续走了多少小道

flag=exhaust-(lastexhaust)\*(lastexhaust)+(lastexhaust+q->RoadLength)\*(lastexhaust+q->RoadLength);

}

if(flag<min)

{

min=flag;

tag=q->RoadKind;

if(tag==1)

{

insertlength=q->RoadLength;

}

(\*key)=q->point;

}

q=q->next;

}

((\*road)->base+(\*road)->length)->point=\*key;

((\*road)->base+(\*road)->length)->exhaust=min;

(\*road)->length++;

if(tag==0)

{

ClearStack(stack);

}

else Push(&stack,insertlength);

break;

}

i++;

p++;

q=p->relate->next;

}

return min;

}

void FindRoad(PList list,int start,int end,int \*totalexhaust,Stack stack,Road \*road)

{

int exhaust;

int key;

if(start!=end)

{

exhaust=Froad(list,start,&key,\*totalexhaust,&stack,&road);

if(key==0)

{

printf("所找寻的两个点不连通\n");

exit(0);

}

\*totalexhaust=exhaust;

FindRoad(list,key,end,totalexhaust,stack,road);

}

}

void OutputRoad(Road road)

{

int i=0;

RoadNode \*p=road.base;

printf("最小疲劳度的路径是\n");

while(i<road.length)

{

printf("%d ",p->point);

p++;

i++;

}

printf("\n");

}

void Result(PList list,int end)

{

Stack stack;

Road road;

int totalexhaust;

InitRoad(&road);

road.base->point=1;

road.base->exhaust=0;

road.length++;

InitStack(&stack);

FindRoad(list,1,end,&totalexhaust,stack,&road);

if(end==n)

{

printf("%d\n",totalexhaust);

OutputRoad(road);

}

else printf("%d->%d %d\n",1,end,totalexhaust);

free(stack.base);

free(road.base);

totalexhaust=0;//为什么totalexhaust不是静态变量还是继承了之前的值??

}

int main()

{

FILE \*input;

PList list;

input=fopen("Road.txt","r");

fscanf(input,"%d%d",&n,&m);

InitList(&list);

InputRelatePoint(&list,input);

Result(list,n);

int i=2;

while(i<list.length)

{

Result(list,i);

i++;

}

return 0;

}

# 六、排序算法比较

6.1 题目简介

利用随机函数产生10个样本，每个样本有20000个随机整数（并使第一个样本是正序，第二个样本是逆序），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。

[基本要求]

（1） 原始数据存在文件中，用相同样本对不同算法进行测试；

（2） 屏幕显示每种排序算法对不同样本所花的时间；

6.2 数据结构

typedef struct node

{

int num;

struct node \*next;

}Node,\*NodeList;

typedef struct key

{

int keynum;

Node \*next;

}Key;

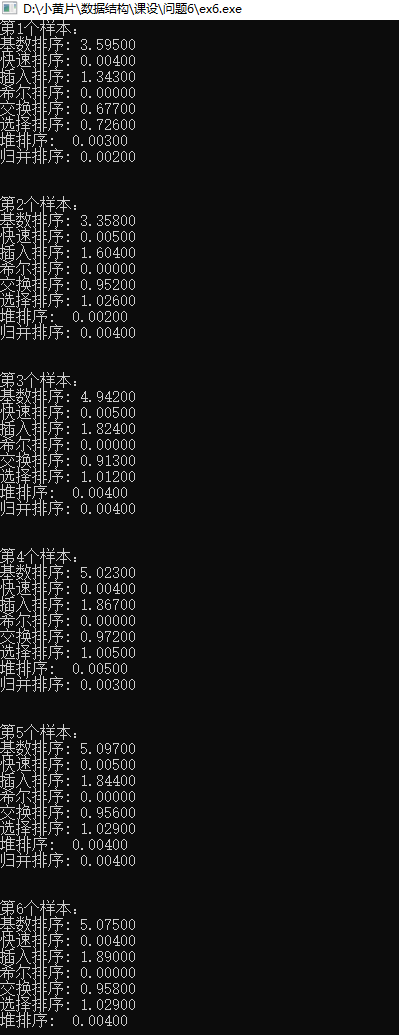
6.3 算法设计思想

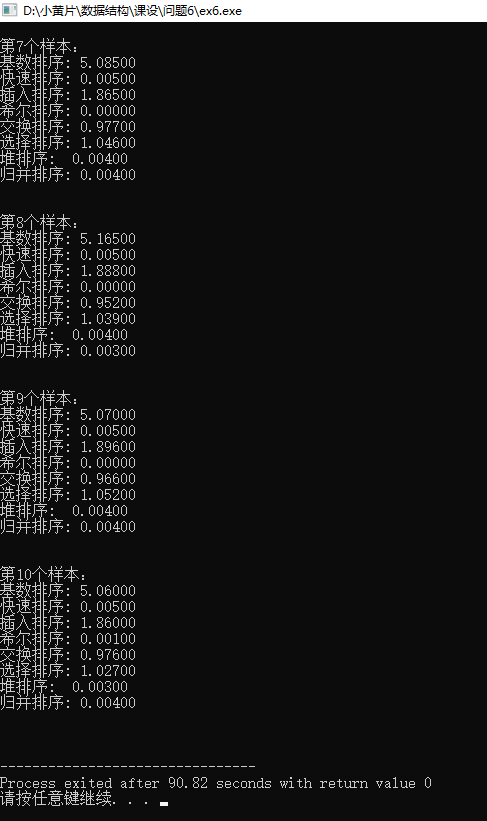
分别有直接插入排序、希尔排序，冒泡排序、快速排序、选择排序、堆排序，归并排序、基数排序

除开基数排序其他都采用数组

基数排序在排序时采用链表

6.4测试数据和结果





6.5算法时间复杂度

直接插入排序O(n2)

希尔排序O(n1.3-2)

冒泡排序O(n2)

快速排序O(nlogn)

选择排序O(n2)

堆排序O(nlogn)

归并排序O(nlogn)

基数排序O(n+d)

6.6源代码

#include<stdio.h>

#include<malloc.h>

#include<stdlib.h>

#include<time.h>

#include<windows.h>

#include<math.h>

typedef struct node

{

int num;

struct node \*next;

}Node,\*NodeList;

typedef struct key

{

int keynum;

Node \*next;

}Key;

void CreateSample(char filename[10][20])

{

int i,j;

FILE \*In[10];

int a[20000];

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

{

In[j]=fopen(filename[j],"w");

if(In[j]==NULL)

{

printf("Can't open the file \n");

exit(0);

}

}

srand((unsigned)time(NULL));

for(j=0;j<10;j+=2)

{

for(i=0;i<20000;i++)

{

a[i]=rand()%20000+1;

if((i+1)%20==0)

fprintf(In[j],"\n");

fprintf(In[j],"%d ",a[i]);

}

for(i=20000-1;i>=0;i--)

{

if((20000-i)%20==0)

fprintf(In[j+1],"\n");

fprintf(In[j+1],"%d ",a[i]);

}

}

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

{

fclose(In[j]);

}

}

void CreateArray(int Randnum[],FILE \*out)

{

int i,n;

for(i=0;i<20000;i++)

{

fscanf(out,"%d",&Randnum[i]);

}

}

void InsertSort(int Randnum[])

{

int i,j;

int flag;

for(i=0;i<20000;i++)

{

flag=Randnum[20000-1];

for(j=19999;j>i;j--)

{

Randnum[j]=Randnum[j-1];

}

for(;j>0;j--)

{

Randnum[j]=Randnum[j-1];

if(flag>Randnum[j])

{

break;

}

}

Randnum[j]=flag;

}

}//插入排序

void ShellInsert(int Randnum[],int dk)

{

int i=dk,j;

int flag;

for(;i<20000;i++)

{

if(Randnum[i]<Randnum[i-dk])

{

flag=Randnum[i];

Randnum[i]=Randnum[i-dk];

for(j=i-dk;j>dk;j-=dk)

{

Randnum[j]=Randnum[j-dk];

if(Randnum[j]<flag)

break;

}

Randnum[j]=flag;

}

}

}

void ShellSort(int Randnum[])

{

int dlta[3]={5,3,1};

int k=0;

for(;k<3;k++)

{

ShellInsert(Randnum,dlta[k]);

}

}

void SwapSort(int Randnum[])

{

int i,j;

int n,t;

n=20000;

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

{

for(j=0;j<n-i-1;j++)

{

if(Randnum[j+1]<Randnum[j])

{

t=Randnum[j+1];

Randnum[j+1]=Randnum[j];

Randnum[j]=t;

}

}

}

}//冒泡排序

void SelectSort(int Randnum[])

{

int i,j;

int t;

int min;

int flag=0;

int n=20000;

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

{

flag=i;

min=Randnum[i];

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

{

if(min>Randnum[j])

{

min=Randnum[j];

flag=j;

}

}

if(flag!=i)

{

t=Randnum[i];

Randnum[i]=Randnum[flag];

Randnum[flag]=t;

}

}

}//选择排序

void Merge(int Randnum[],int a[],int i,int m,int n)

{

int j,k;

int s=0;

for(j=m+1,k=i;i<=m&&j<=n;k++)

{

if(Randnum[i]<Randnum[j]){

a[k]=Randnum[i];

i++;

}

else{

a[k]=Randnum[j];

j++;

}

}

if(i<=m)

{

while(i<=m)

{

a[k]=Randnum[i];

k++;

i++;

}

}

if(j<=n)

{

while(j<=n)

{

a[k]=Randnum[j];

k++;

j++;

}

}

}

void Output(int a[])

{

int i=0;

for(i=0;i<20000;i++)

{

printf("%d ",a[i]);

if((i+1)%20==0) printf("\n");

}

}

void MSort(int Randnum[],int a[],int k)

{

int i=0,j;

while(i<=20000-2\*k)

{

Merge(Randnum,a,i,i+k-1,i+2\*k-1);

i+=2\*k;

}

if(i<20000-k+1)

{

Merge(Randnum,a,i,i+k-1,20000-1);

}

else

{

for(j=i;j<20000;j++)

{

a[j]=Randnum[j];

}

}

for(i=0;i<20000;i++)

{

Randnum[i]=a[i];

}

}

void MergeSort(int Randnum[])

{

int a[20000];

int m=1;

while(m<20000)

{

MSort(Randnum,a,m);

m\*=2;

}

}//归并排序

int Partition(int Randnum[],int low,int high)

{

int t,i,j,key;

i=low;

j=high;

key=Randnum[low];

while(i<j)

{

while(i<j&&Randnum[j]>=key) j--;

Randnum[i]=Randnum[j];

while(i<j&&Randnum[i]<=key) i++;

Randnum[j]=Randnum[i];

}

Randnum[i]=key;

return i;

}

void QSort(int Randnum[],int low,int high)

{

int key;

if(low<high)

{

key=Partition(Randnum,low,high);

QSort(Randnum,low,key-1);

QSort(Randnum,key+1,high);

}

}//快速排序

void HeapAdjust(int Randnum[],int s,int m)

{

int flag;

flag=Randnum[s-1];

int j;

for(j=2\*s;j-1<m;j\*=2)

{

if(j<m&&Randnum[j-1]>Randnum[j]) j++;

if(flag<=Randnum[j-1]) break;

Randnum[s-1]=Randnum[j-1];

s=j;

}

Randnum[s-1]=flag;

}

void HeapSort(int Randnum[])

{

int i;

int t;

for(i=20000/2;i>0;i--)

{

HeapAdjust(Randnum,i,20000);

}

for(i=19999;i>0;i--)

{

t=Randnum[i];

Randnum[i]=Randnum[0];

Randnum[0]=t;

HeapAdjust(Randnum,1,i-1);

}

}//堆排序

NodeList InitNodelist(NodeList L)

{

L=(NodeList)malloc(sizeof(Node));

L->next=NULL;

return L;

}

NodeList CreateNodelist(NodeList L,int Randnum[])

{

int i=0;

Node \*p,\*q;

q=L;

while(i<20000)

{

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

p->num=Randnum[i];

i++;

q->next=p;

q=p;

}

q->next=NULL;

return L;

}

int High(int Randnum[])

{

int i=0;

int max=0;

int flag,tag=0;

while(i<20000)

{

flag=Randnum[i];

while(flag!=0)

{

tag++;

flag=flag/10;

}

if(tag>max)

{

max=tag;

}

tag=0;

i++;

}

return max;

}

NodeList Distribute(NodeList L,Key f[],Key e[],int i)

{

Node \*p1,\*p2,\*q1,\*q2;

p1=L->next;

p2=p1->next;

int flag;

while(p1)

{

if(i!=0)

flag=(int)((p1->num)/(pow(10,i)))%10;

else flag=(p1->num)%10;

q2=f[flag].next;

if(q2==NULL)

{

p1->next=q2;

f[flag].next=p1;

}

else

{

while(q2)

{

q1=q2;

q2=q1->next;

}

p1->next=q2;

q1->next=p1;

}

e[flag].next=p1;

p1=p2;

if(p2!=NULL)

{

p2=p1->next;

}

}

L->next=NULL;

return L;

}

NodeList Collect(NodeList L,Key f[],Key e[])

{

int i;

Node \*p1,\*p2,\*q;

q=L;

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

{

p1=f[i].next;

if(p1==NULL)

continue;

q->next=p1;

p2=p1->next;

while(p2)

{

p1=p2;

p2=p1->next;

}

q=p1;

f[i].next=NULL;

e[i].next=NULL;

}

q->next=NULL;

return L;

}

double RadixSort(int Randnum[])

{

time\_t time\_start;

time\_t time\_over;

Key e[10];

Key f[10];

NodeList L;

int i=0,n;

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

{

e[i].keynum=i;

f[i].keynum=i;

f[i].next=NULL;

e[i].next=NULL;

}

L=InitNodelist(L);

L=CreateNodelist(L,Randnum);

n=High(Randnum);

time\_start=clock();

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

{

L=Distribute(L,f,e,i);

L=Collect(L,f,e);

}

time\_over=clock();

return (double)(time\_over-time\_start)/CLOCKS\_PER\_SEC;

}

void Output(int Randnum[],char filename[10][20])

{

time\_t time\_start;

time\_t time\_over;

FILE \*In[10];

int i=0;

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

{

printf("第%d个样本：\n",i+1);

In[i]=fopen(filename[i],"r");

CreateArray(Randnum,In[i]);

double run\_time;

run\_time=RadixSort(Randnum);

printf("基数排序: %.5lf\n",run\_time);

CreateArray(Randnum,In[i]);

time\_start=clock();

QSort(Randnum,0,19999);

time\_over=clock();

printf("快速排序: %.5lf\n",(double)(time\_over-time\_start)/CLOCKS\_PER\_SEC);

CreateArray(Randnum,In[i]);

time\_start=clock();

InsertSort(Randnum);

time\_over=clock();

printf("插入排序: %.5lf\n",(double)(time\_over-time\_start)/CLOCKS\_PER\_SEC);

CreateArray(Randnum,In[i]);

time\_start=clock();

ShellSort(Randnum);

time\_over=clock();

printf("希尔排序: %.5lf\n",(double)(time\_over-time\_start)/CLOCKS\_PER\_SEC);

CreateArray(Randnum,In[i]);

time\_start=clock();

SwapSort(Randnum);

time\_over=clock();

printf("交换排序: %.5lf\n",(double)(time\_over-time\_start)/CLOCKS\_PER\_SEC);

CreateArray(Randnum,In[i]);

time\_start=clock();

SelectSort(Randnum);

time\_over=clock();

printf("选择排序: %.5lf\n",(double)(time\_over-time\_start)/CLOCKS\_PER\_SEC);

CreateArray(Randnum,In[i]);

time\_start=clock();

HeapSort(Randnum);

time\_over=clock();

printf("堆排序: %.5lf\n",(double)(time\_over-time\_start)/CLOCKS\_PER\_SEC);

CreateArray(Randnum,In[i]);

time\_start=clock();

MergeSort(Randnum);

time\_over=clock();

printf("归并排序: %.5lf\n",(double)(time\_over-time\_start)/CLOCKS\_PER\_SEC);

printf("\n\n");

}

}

int main()

{

int Randnum[20000];

char filename[10][20]={"Sample1.txt","Sample2.txt","Sample3.txt","Sample4.txt","Sample5.txt","Sample6.txt","Sample7.txt","Sample8.txt","Sample9.txt","Sample10.txt"};

CreateSample(filename);

Output(Randnum,filename);

return 0;

}

# 七、朋友圈

7.1 题目简介

某学校有N个学生，形成M个俱乐部。每个俱乐部里的学生有着相似的兴趣爱好，形成一个朋友圈。一个学生可以同时属于若干个不同的俱乐部。根据“我的朋友的朋友也是我的朋友”这个推论可以得出，如果A和B是朋友，且B和C是朋友，则A和C也是朋友。请编写程序计算最大朋友圈中有多少人。

[基本要求]

（1）输入说明：输入的第一行包含两个正整数N (N<=30 000)和M (M<=1000)，分别代表学校的学生总数和俱乐部的个数。随后的M行每行按以下格式给出一个俱乐部的信息，其中学生从1-N编号：

第i个俱乐部的人数Mi（空格）学生1（空格）学生2… 学生Mi

（2）输出说明：输出一个整数，表示在最大朋友圈中有多少人。

（3）测试用例：

输入 7 4

3 1 2 3

2 1 4

3 5 6 7

1 6

输出 4

7.2 数据结构

typedef struct Clubfriend

{

int keynum;

struct Clubfriend \*next;

}Clubfriend;

typedef struct

{

int key;

Clubfriend \*next;

}Student;

typedef struct

{

Student \*base;

int length;

}StudentsList;

typedef struct

{

int \*studentNum;

int length;

}StudentNode;

typedef struct

{

int club;

StudentNode clubStudent;

}Club;

typedef struct

{

Club \*base;

int length;

}ClubList;

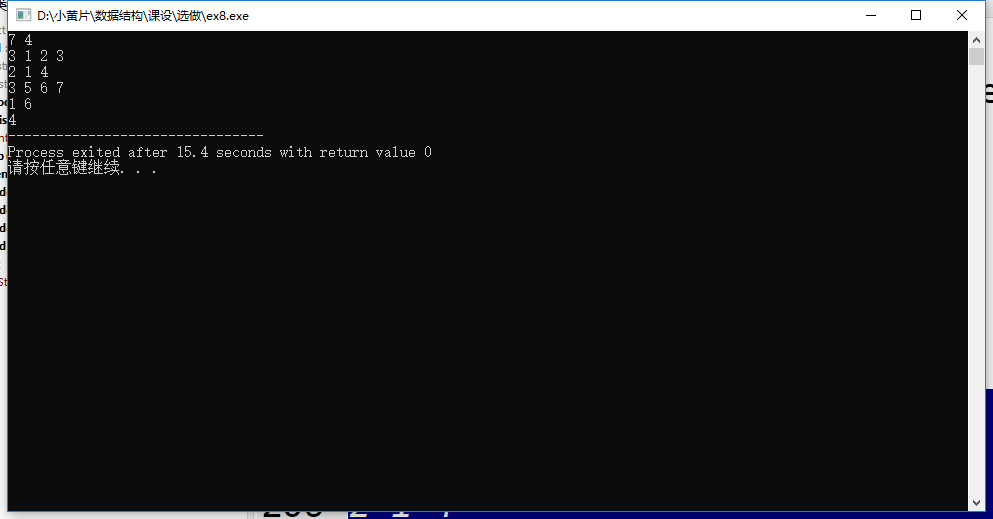
7.3 算法设计思想

首先存储一个所有的club的信息，然后将所有的club的同样的朋友存储在对应的同学的节点中。

然后对应每个节点的朋友，找到对应该朋友的节点，来增添当前节点的新朋友。

最后遍历一边所有节点，输出最大的朋友圈

7.4测试数据和结果

7.5算法时间复杂度

CreateStudentList(StudentsList \*list,ClubList clublist)

时间复杂度O(n);

Add(StudentsList \*list,int i,int e)

时间复杂度O(n)

CreateFriends(Student \*student,ClubList clublist)

时间复杂度O(n2)

7.6源代码

#include<stdio.h>

#include<malloc.h>

#include<stdlib.h>

int N,M;

typedef struct Clubfriend

{

int keynum;

struct Clubfriend \*next;

}Clubfriend;

typedef struct

{

int key;

Clubfriend \*next;

}Student;

typedef struct

{

Student \*base;

int length;

}StudentsList;

typedef struct

{

int \*studentNum;

int length;

}StudentNode;

typedef struct

{

int club;

StudentNode clubStudent;

}Club;

typedef struct

{

Club \*base;

int length;

}ClubList;

void CreateStudentNode(StudentNode \*studentnode,int nodelength)

{

int i=0;

studentnode->studentNum=(int\*)malloc(nodelength\*sizeof(int));

while(i<nodelength)

{

scanf("%d",studentnode->studentNum+i);

i++;

}

studentnode->length=nodelength;

}

void CreateClub(ClubList \*clublist)

{

clublist->base=(Club\*)malloc(M\*sizeof(Club));

int i=0,j;

int nodelength;

while(i<M)

{

clublist->base[i].club=i+1;

scanf("%d",&nodelength);

CreateStudentNode(&(clublist->base[i].clubStudent),nodelength);

i++;

}

clublist->length=M;

}

int ExistInStudent(Student m,int e)

{

Clubfriend \*p;

p=m.next;

while(p)

{

if(p->keynum==e)

return 1;

p=p->next;

}

return 0;

}

void CreateFriends(Student \*student,ClubList clublist)

{

int i,k,j=0;

StudentNode p;

Clubfriend \*q;

for(;j<clublist.length;j++)

{

p=clublist.base[j].clubStudent;

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

{

if(p.studentNum[i]==student->key)

{

break;

}

}

if(i!=clublist.base[j].clubStudent.length)

{

k=0;

while(k<clublist.base[j].clubStudent.length)

{

if(k==i||ExistInStudent(\*student,p.studentNum[k]))

{

k++;

continue;

}

q=(Clubfriend\*)malloc(sizeof(Clubfriend));

q->keynum=\*(clublist.base[j].clubStudent.studentNum+k);

q->next=student->next;

student->next=q;

k++;

}

}

}

}

StudentsList \*Add(StudentsList \*list,int i,int e)

{

Student s=list->base[e-1];

Clubfriend \*p,\*q;

p=s.next;

while(p)

{

if(p->keynum==list->base[i].key||ExistInStudent(list->base[i],p->keynum))

{

p=p->next;

continue;

}//如果存在这个朋友或者这个num等于自己，跳过

q=(Clubfriend\*)malloc(sizeof(Clubfriend));

q->keynum=p->keynum;

q->next=list->base[i].next;

list->base[i].next=q;

p=p->next;

}

return list;

}

StudentsList \*Extrafriend(StudentsList \*list,int i)

{

Clubfriend \*p;

p=list->base[i].next;

while(p)

{

list=Add(list,i,p->keynum);

p=p->next;

}

return list;

}

void CreateStudentList(StudentsList \*list,ClubList clublist)

{

int num,max=0;

list->base=(Student\*)malloc(N\*sizeof(Student));

int i=0;

while(i<N)

{

list->base[i].key=i+1;

list->base[i].next=(Clubfriend\*)malloc(sizeof(Clubfriend));

list->base[i].next=NULL;

i++;

}

list->length=N;

for(i=0;i<list->length;i++)

{

CreateFriends(&list->base[i],clublist);

}

for(i=0;i<list->length;i++)

{

list=Extrafriend(list,i);

}

}

int Traverse(Student m)

{

Clubfriend \*p;

int i=0;

p=m.next;

while(p)

{

i++;

p=p->next;

}

return i;

}

int main()

{

ClubList clublist;

StudentsList list;

scanf("%d%d",&N,&M);

CreateClub(&clublist);

CreateStudentList(&list,clublist);

int i=0,max=0,nums;

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

{

nums=Traverse(list.base[i])+1;//加上自己

if(nums>max) max=nums;

}

printf("%d",max);

}

/\*

7 4

3 1 2 3

2 1 4

3 5 6 7

1 6

7 4

4 1 2 3 5

2 1 4

3 5 6 7

1 6

\*/

# 八、社交网络图中结点的“重要性”计算

8.1 题目简介

在社交网络中，个人或单位（结点）之间通过某些关系（边）联系起来。他们受到这些关系的影响，这种影响可以理解为网络中相互连接的结点之间蔓延的一种相互作用，可以增强也可以减弱。而结点根据其所处的位置不同，在网络中体现的重要性也不尽相同。

“紧密度中心性”是用来衡量一个结点达到其他结点的“快慢”的指标，即一个有较高中心性的结点比有较低中心性的结点能够更快地（平均意义下）到达网络中的其他结点，因而在该网络的传播过程中有更重要的价值。在有N个结点的网络中，结点vi的“紧密度中心性” 数学上定义为 到其余所有结点(j!=i)的最短距离的平均值的倒数：



对于非连通图，所有结点的紧密度中心性都是0。

本实验给定一个无权的无向图以及其中的一组结点，要求计算这组结点中每个结点的紧密度中心性。

[基本要求]

（1）输入说明：输入的第一行给出两个正整数N (N<=1000)和M，其中N是图中结点个数，结点编号从1到N； M（M<=10000）是边的数目。随后的M行中，每行给出一条边的信息，即该边连接的两个结点编号，中间用空格分隔。最后一行给出需要计算紧密度中心性的这组结点的个数K（K<=100），以及K个结点的编号，用空格分隔。

（2）输出说明：按照“Cc(i)=x.xx”的格式输出K个给定结点的紧密度中心性，每个输出占一行，结果精确到小数点后2位。

（3）测试用例：

输入 5 8

1 2

1 3

1 4

2 3

3 4

4 5

2 5

3 5

2 4 3

输出 Cc(4) = 0.80

Cc(3) = 1.00

8.2 数据结构

typedef struct Side

{

int num;

struct Side \*next;

}Side;

typedef struct Node

{

int keynum;

Side \*next;

}Node;

typedef struct

{

Node \*base;

int length;

}NodeList;

typedef struct

{

int \*foundnum;

int length;

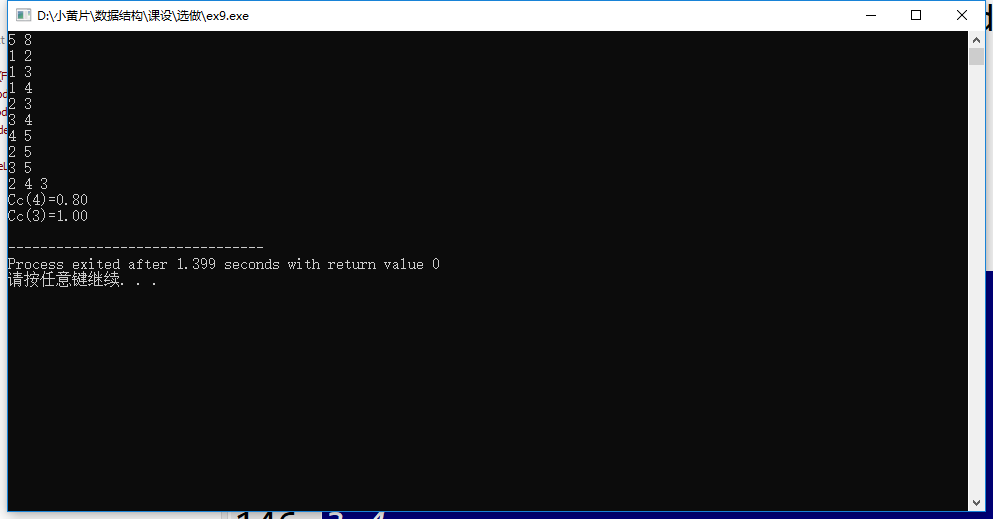
}FindList;

8.3 算法设计思想

建立一个邻接表，邻接表建立的头建立一个1，2，3，4，5…的顺序表，每个节点要接到与其相邻的边。

在寻找的时候，寻找到对应的点，到其他剩余的点最短距离。

8.4测试数据和结果



8.5算法时间复杂度

FindLength(NodeList nodelist,int i,int target)

时间复杂度O(n2)

8.6源代码

#include<stdio.h>

#include<malloc.h>

#include<stdlib.h>

int N,M,K;

typedef struct Side

{

int num;

struct Side \*next;

}Side;

typedef struct Node

{

int keynum;

Side \*next;

}Node;

typedef struct

{

Node \*base;

int length;

}NodeList;

typedef struct

{

int \*foundnum;

int length;

}FindList;

void CreateNode(NodeList \*nodelist)

{

nodelist->base=(Node\*)malloc(N\*sizeof(Node));

int i=0;

while(i<N)

{

nodelist->base[i].keynum=i+1;

nodelist->base[i].next=(Side\*)malloc(sizeof(Side));

nodelist->base[i].next=NULL;

i++;

}

nodelist->length=N;

}

void CreateSides(NodeList \*nodelist)

{

int j=0;

int begin,end,sidelength;

Side \*p,\*q;

while(j<M)

{

scanf("%d%d",&begin,&end);

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

p->num=end;

p->next=nodelist->base[begin-1].next;

nodelist->base[begin-1].next=p;

q=(Side\*)malloc(sizeof(Side));

q->num=begin;

q->next=nodelist->base[end-1].next;

nodelist->base[end-1].next=q;

j++;

}

}

void CreateFindList(FindList \*findlist)

{

int s=0;

findlist->foundnum=(int\*)malloc(K\*sizeof(int));

while(s<K)

{

scanf("%d",(findlist->foundnum+s));

s++;

}

findlist->length=K;

}

int FindLength(NodeList nodelist,int i,int target)

{

int length=0;

int min\_length=N-1;

Side \*p;

p=nodelist.base[target].next;

while(p)

{

if(p->num==i+1)

{

min\_length=1;

break;

}

p=p->next;

}//如果在该结点中能够找到则返回1

if(!p)

{

p=nodelist.base[target].next;

while(p)

{

length=FindLength(nodelist,i,p->num-1);

if(min\_length>length+1) min\_length=length+1;

p=p->next;

}

}//如果找不到就对target的每个连接的点进行寻找，最后返回length

return min\_length;

}

void OutResult(NodeList \*nodelist,int value)

{

int target=value-1;

int TotalLength=0;

int i=0;

while(i<nodelist->length)

{

if(i==target)

{

i++;

continue;

}

TotalLength+=FindLength(\*nodelist,i,target);

i++;

}

printf("Cc(%d)=%.2f\n",value,(float)(N-1)/TotalLength);

}

int main()

{

NodeList nodelist;

FindList findlist;

scanf("%d%d",&N,&M);

CreateNode(&nodelist);

CreateSides(&nodelist);

scanf("%d",&K);

CreateFindList(&findlist);

int find=0;

while(find<K)

{

OutResult(&nodelist,findlist.foundnum[find]);

find++;

}

return 0;

}

/\*

5 8

1 2

1 3

1 4

2 3

3 4

4 5

2 5

3 5

2 4 3

\*/

# 九、魔法优惠劵

9.1 题目简介

在火星上有个魔法商店，通过魔法优惠券。每个优惠券上印有一个整数面值K，表示若你在购买某商品使用这张优惠券，可以得到K倍该商品价值的回报。该商店还免费赠送一些有价值的商品，但如果你在领取免费赠品的时候使用面值为正的优惠券，则必须倒贴给商品K倍该商品价值的金额……但是不要紧，还有面值为负的优惠券可以用。

例如，给定一组优惠券，面值分别为1、2、4、-1；对应一组商品，价值为火星币7、6、-2、-3，其中负的价值表示该商品是免费赠品。我们可以将优惠券3（面值4）用在商品1（价值7）上，得到火星币28的回报。优惠券4（面值-1）用在商品4（价值-3）上，得到火星币3的回报。但是，如果一不小心把优惠券3（面值4）用到商品4（价值-3）上，你必须倒贴给商店火星币12个。同样，把优惠券4（面值-1）用到商品1（价值7）上，你必须倒贴给商店火星币7个。

规定每张优惠券和每件商品都只能最多被使用一次，求你可以得到的最大回报。

[基本要求]

（1）输入说明：输入有两行。第一行首先给出优惠券的个数N，随后给出N个优惠券的整数面值。 第二行首先给出商品的个数M，随后给出M个商品的整数价值。N和M在[1,106]之间，所有的数据大小不超过230，数字间以空格分隔。

（2）输出说明：输出可以得到的最大回报。

（3）测试用例：

输入 4 1 2 4 -1 // -1 1 2 4

4 7 6 -2 -3 // -3 -2 6 7

输出 43

输入：4 3 2 6 1 // 1 2 3 6

3 2 6 3 // 2 3 6

输出：49

输入： 7 3 36 -1 73 2 3 6 // -1 2 3 3 6 36 73

6 -1 -1 -1 -1 -1 -1 // -1 -1 -1 -1 -1 -1

输出： 1

9.2 数据结构

typedef struct Node

{

int value;

bool used;

}Node;

typedef struct List

{

Node \*elem;

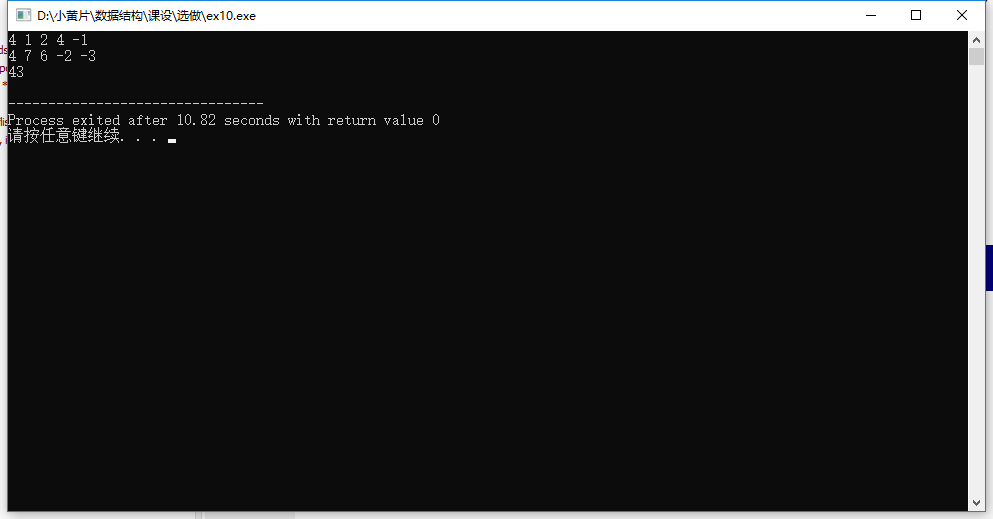
int length;

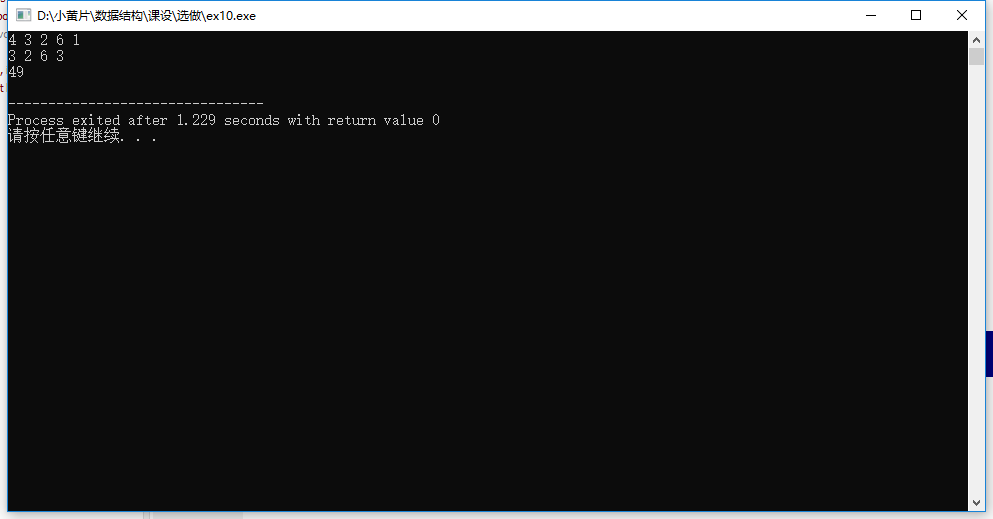
}Goods,Coupons;

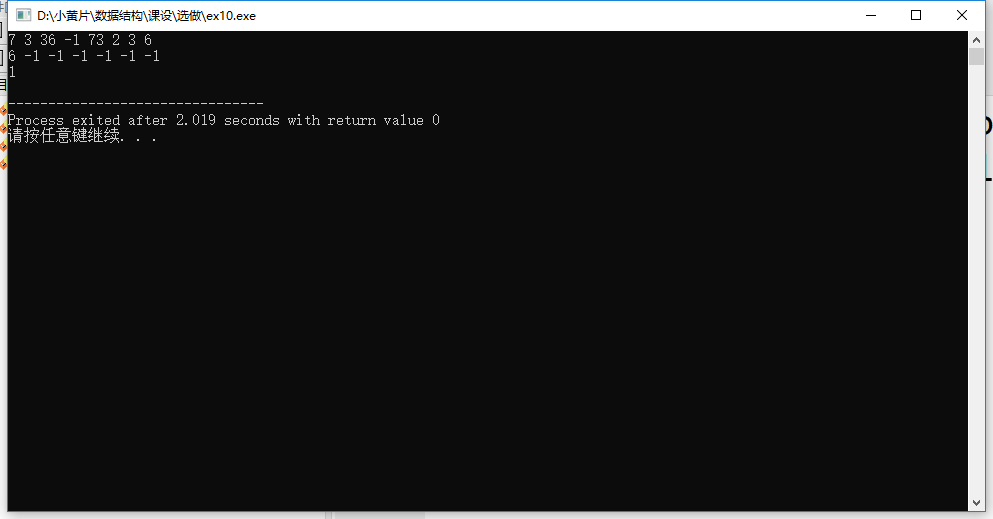
9.3 算法设计思想

对商品的价格和优惠券进行排序，从小到大，一一进行比较，从大的值开始进行比较，如果商品价值和优惠券的值相乘如果大于零，则可以相加。

9.4测试数据和结果







9.5算法时间复杂度

Calculate(Goods goods,Coupons coupons)

时间复杂度O(n\*m)

9.6源代码

#include<stdio.h>

#include<malloc.h>

#include<stdlib.h>

int N,M;

typedef struct Node

{

int value;

bool used;

}Node;

typedef struct List

{

Node \*elem;

int length;

}Goods,Coupons;

void CreateGoods(Goods \*goods)

{

goods->elem=(Node\*)malloc(N\*sizeof(Node));

if(!goods->elem)

{

printf("Create Goods fail!\n");

exit(0);

}

int i=0;

while(i<N)

{

scanf("%d",&goods->elem[i].value);

goods->elem[i].used=0;

i++;

}

goods->length=N;

}

void CreateCoupons(Coupons \*coupons)

{

coupons->elem=(Node\*)malloc(M\*sizeof(Node));

if(!coupons->elem)

{

printf("Create Coupons fail!\n");

exit(0);

}

int i=0;

while(i<M)

{

scanf("%d",&coupons->elem[i].value);

coupons->elem[i].used=0;

i++;

}

coupons->length=M;

}

int Partition(struct List \*list,int low,int high)

{

int t,i,j,key;

i=low;

j=high;

key=list->elem[low].value;

while(i<j)

{

while(i<j&&list->elem[j].value>=key) j--;

list->elem[i].value=list->elem[j].value;

while(i<j&&list->elem[i].value<=key) i++;

list->elem[j].value=list->elem[i].value;

}

list->elem[i].value=key;

return i;

}

void QSort(struct List \*list,int low,int high)

{

int key;

if(low<high)

{

key=Partition(list,low,high);

QSort(list,low,key-1);

QSort(list,key+1,high);

}

}

int Calculate(Goods goods,Coupons coupons)

{

int i=goods.length-1,j=coupons.length-1;

int max=0;

int total,result=0;

int goodsflag=-1,couponsflag=-1;

while(i>=0)

{

while(j>=0)

{

if(goods.elem[i].used!=1&&coupons.elem[j].used!=1&&(goods.elem[i].value)\*(coupons.elem[i].value)>0)

{

total=(goods.elem[i].value)\*(coupons.elem[j].value);

if(max<total)

{

max=total;

goodsflag=i;

couponsflag=j;

}

}

j--;

}

if(goodsflag!=-1)

{

goods.elem[goodsflag].used=1;

coupons.elem[couponsflag].used=1;

result+=max;

}

i--;

j=coupons.length-1;

goodsflag=-1;

couponsflag=-1;

max=0;

}

return result;

}

int main()

{

Goods goods;

Coupons coupons;

int Max\_result;

scanf("%d",&N);

CreateGoods(&goods);

scanf("%d",&M);

CreateCoupons(&coupons);

QSort(&goods,0,N-1);

QSort(&coupons,0,M-1);//进行快速排序

Max\_result=Calculate(goods,coupons);

printf("%d\n",Max\_result);

return 0;

}

/\*

4 1 2 4 -1

4 7 6 -2 -3

4 3 2 6 1

3 2 6 3

7 3 36 -1 73 2 3 6

6 -1 -1 -1 -1 -1 -1

\*/

# 十、课程设计总结

10.1完成情况（代码行数）

必做题1：537行

必做题2：296行

必做题3：300行

必做题4：406行

必做题5：308行

必做题6：505行

选做题8：201行

选做题9：139行

选做题10：134行

代码行数共：2826行

[**10.2 心得体会**](#_Toc6584)

在写这次课设的时候，其实因为自己时转专业的，所以补选了很多课程，所以写课设的时间并没有太长。也就是并没有从老师将所有课设的题目告知我们就开始进行课设的编程。

但其实我认为在整个学期的编程中，有八周都是要交程序和程序的报告，感觉每一周都需要紧迫感，有一段时间自己是早上5点多起床，跑跑步，吃完早饭，如果没有课就去图书馆，然后带上自己很重的电脑，开始敲代码。其实要说很累，的确是很累的，不过当我选择来这个专业，就要做好一个完全的心理准备。在任何一个专业都不是能够轻松度过的，我来这个专业是的的确确想要学到实实在在的东西。

然而数据结构这门课程让我体会到计算机的确是有很多东西能学的，像我在大一在其他专业学了C语言，我还清晰记得，自己在大一的时候写C语言的课设时，也就是一个程序量并不大的图书管理系统，在写这个程序的时候自己觉得大一的时候的确是有点困难的，但我在学了数据结构之后，在掌握到其实有很多算法我还并不了解，在了解之后，我发觉其实大一的图书管理系统并不是那样困难的。相比较起来，图书管理系统的代码量也并不是很多，但在完成课设之后，我认为代码量的确是很重要的，就像老师说的那样，代码量的确会提升编码能力，在写每个题之前都好好打一个草稿，需要运用到什么数据结构，运用什么算法来进行编码。而且通过这次的课设的设计，我也从很多同学身上学到了很多东西，比如要学会代码的格式要好好排布，而且对注释要尽可能的详细不复杂。

其实对于这门课程，我认为是有感激之情的，每次上课的时候都要认真听讲，感觉老师没说的一个字都是必须要好好学会的东西。起初的时候觉得老师的语速很快，而且觉得老师很严格，起初的时候是有点害怕的，但是到现在，我认为老师其实并没有那么严格，在上课的时候总是会用一些同学的例子告诉我们一些该避开的要素，课堂气氛有的时候并没有那么活跃，但是在大部分的时候的确是很活跃的。

最后，通过一整个学期的学习，我觉得从数据结构的课程开始，就非常的充实，最后要谢谢老师，谢谢同学们。