



## 数据结构课程设计2022

## ——各模块设计报告

小组成员： 韩旭、陈可儿、齐雨婷

学 院 ：计算机学院（国家示范性软件学院）

专 业 ： 计算机科学与技术

班 级 ： 2020211307

指导教师： 张海旸老师

模块一：课程课外信息管理和查询

1. **排序**

快速排序---挖坑版

性能：时间复杂度O(Nlog2N)，空间复杂度O(log2N)

优点：平均情况下比较快，不需要额外空间，优化后使用栈不需要递归。

缺点：属于不稳定排序，最坏的情况的情况下时间复杂度为O(N^2)

**基本思想：**

1.对记录中time字段的处理：因为涉及到对数据库中记录的time字段进行排序，而这个字段是以QString即一个字符串为整体存储的，所以首先要做的是对time字段按照周次、小时分别存储，这里采用的做法是构造一个结构体，成员分别为id,week,hour,id存放某条记录的id,week存放该记录time字段中的周次，hour存放记录中time字段的小时部分。申请该结构体类型的数组，遍历数据库，用数组存放每条记录的id，time字段中的周次和小时部分。

1. 快速排序：

①单次排序：挖坑法，定义两个int型变量left,right，left存放起始位置，right存放最后一个元素的位置，然后指定一个基数key（这里选的是left）,作为坑，right寻找比基数（key）小的数字，找到将right的数据赋给left，right成为一个新坑，然后left寻找比基数（key）大的数字，找到后将left的数据赋给right，left成为一个坑，循环这个过程，直到begin与end相遇，然后将key返回给那个坑，最后使得序列中key的左边都是比key小的数，key的右边都是比key大的数。单次结束后，返回最后坑的位置，作为下次单排的中轴数索引。

②使用栈来存储子序列的起点后终点，如果中轴数索引两个分区存在,则将起点和终点入栈。当栈不为空的时候，从栈里面取出序列的起始点并利用单词排序返回该序列中轴数的索引，再将新的序列区间入栈，如此循环下去。

**比较过程：**

while (left < right)

{

//right向左，找小

while (left < right&&(a[right].week > key\_w||(a[right].week == key\_w&&a[right].hour>key\_h)))//week优先于hour

{

right--;

}

//填坑

a[left].id = a[right].id;

a[left].week = a[right].week;

a[left].hour = a[right].hour;

//left向右，找大

while (left < right&&(a[left].week < key\_w||(a[left].week == key\_w&&a[left].hour < key\_h)))

{

left++;

}

//填坑

a[right].id = a[left].id;

a[right].week = a[left].week;

a[right].hour = a[left].hour;

}

1. **解压缩**

哈夫曼压缩：

**基本思想：**

①压缩部分：

1.读取传入文件，进行字符权重统计

2.将出现的字符放入哈夫曼树结点，构建哈夫曼树，获取哈夫曼编码

3.将编码相关信息写入压缩后的文件，再将传入文件的每个字符按照哈夫曼编码转换，每8个二进制位的字符串转化为ASCII码，传入压缩后的文件

②解压部分：

1.将传入的已压缩文件进行文件流读取，获取编码信息进行还原

**具体思路：**

读取传入文件：使用Qt自带的QFile文件流配以QDatastream辅助。

权重统计：用map记录：map<unsigned char ,int>Weightmap。

哈夫曼结点的构造：

struct **HuffmanTreeNode**

{

pair<unsigned char, int>Weight; //记录字符和其权重

string zippassword; //等待获取的哈夫曼编码

HuffmanTreeNode\* leftson; //左儿子指针

HuffmanTreeNode\* rightson; //右儿子指针

bool tag; //是否为叶子结点的标记，true为非叶子结点

};

//哈夫曼树结点

构建哈夫曼树：

将建立好的结点存放到一个vector内（vector<HuffmanTreeNode\* > container），进入循环：根据哈夫曼结点的权重比较进行排序（直接调用sort），每次将两个最小的结点权重取出，相加得到权重和，以权重和建立一个新的结点，新结点的左右孩子结点就是这两个结点，将新结点加入到vector中，那两个结点删除。循环结束条件为vector内只剩下一个结点，这个结点便是哈夫曼树的根结点。

获取哈夫曼编码：

从根结点出发遍历，左子树的string+“0”，右子树的string+“1”。并将unsigned char型的字符与这个string用map<unsigned char,string> PasswordMap 记录。

传入辅助信息：

我们传入的信息有，PasswordMap.size()，循环传入PasswordMap->first（字符）,PasswordMap->second.size()（记录编码长度）,PasswordMap->second（这个字符串有可能超过8位，每个字节不足8位的部分补0）直到完全传入。

传入哈夫曼编码阶段：

遍历先前在读取文件时获取的字符串，将字符串的每一位转化为哈夫曼编码，用另一个字符串储存。这里不妨称这个字符串为二进制字符串，这个二进制字符串按8分割到最后可能会有不足8位的部分，对其进行补0操作。最后再传入一个字节，记录补0数。

解压部分：对应上部分的操作进行对应读取，主要思路是将压缩时传入的辅助信息提取，建立一个新的Map映射，根据这个映射，将原文件还原。

1. **查找**

**基本思想：**

根据课程表名称的首字母或者首字符创建哈希表，然后根据输入课程名称首字母或者首字符，利用哈希查找到对应表项再利用KMP算法逐个匹配。

**核心算法：**

//创建哈希表

void **CreateHashTable**(Point \*arr[],const char \*src, QList<EventInfo> lEvents)

{

if(src == NULL) return;

//初始化数组

int i;

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

arr[i] = NULL;

i = 0;

Point \*pTemp = NULL;//记录新添加的节点

Point \*pMark = NULL;//标记添加位置

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

{

//向Hash\_Table添加元素

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

pTemp->pNext = NULL;

pTemp->nValue = i;

string s = lEvents[i].name.toStdString();

char \* strs = new char[s.length() + 1];

strcpy(strs, s.c\_str());

strcpy(pTemp->Name, strs);

if(arr[(int)src[i]] == NULL)

{

arr[(int)src[i]] = pTemp;

}

else

{

pMark = arr[(int)src[i]];

//找到要插入链表位置的尾，尾添加

while(pMark->pNext != NULL)

pMark = pMark->pNext;

pMark->pNext = pTemp;

}

++i;

}

}

1. **时间冲突检测**

**基本思想：**

将用户输入的活动开始时间分周次和时间分别存储，读取数据库每一项记录，将time字段分周次和时间存，最后进行比较活动开始时间是否落在某一记录的时间区间中，若是，则弹窗提示。

**核心算法：**

存放数据库记录中的time字段

void TimeSort::**init**(TimeInfo t[], QList<EventInfo> lEvents)

{

for(int i=0;i<lEvents.size();i++)

{

t[i].id=i;

//存周次

if(lEvents[i].time.mid(2,1)=="一")

t[i].week=1;

else if(lEvents[i].time.mid(2,1)=="二")

t[i].week=2;

else if(lEvents[i].time.mid(2,1)=="三")

t[i].week=3;

else if(lEvents[i].time.mid(2,1)=="四")

t[i].week=4;

else if(lEvents[i].time.mid(2,1)=="五")

t[i].week=5;

else if(lEvents[i].time.mid(2,1)=="六")

t[i].week=6;

else if(lEvents[i].time.mid(2,1)=="日")

t[i].week=7;

else

{

if(lEvents[i].time.mid(1,1)=="二")

t[i].week=2;

else

t[i].week=6;

}

//存时

if(lEvents[i].time.mid(3,2)=="08")

t[i].hour=8;

else if(lEvents[i].time.mid(3,2)=="09")

t[i].hour=9;

else if(lEvents[i].time.mid(3,2)=="10")

t[i].hour=10;

else if(lEvents[i].time.mid(3,2)=="11")

t[i].hour=11;

else if(lEvents[i].time.mid(3,2)=="12")

t[i].hour=12;

else if(lEvents[i].time.mid(3,2)=="13")

t[i].hour=13;

else if(lEvents[i].time.mid(3,2)=="14")

t[i].hour=14;

else if(lEvents[i].time.mid(3,2)=="15")

t[i].hour=15;

else if(lEvents[i].time.mid(3,2)=="16")

t[i].hour=16;

else if(lEvents[i].time.mid(3,2)=="17")

t[i].hour=17;

else if(lEvents[i].time.mid(3,2)=="18")

t[i].hour=18;

else if(lEvents[i].time.mid(3,2)=="19")

t[i].hour=19;

else if(lEvents[i].time.mid(3,2)=="20")

t[i].hour=20;

else if(lEvents[i].time.mid(3,2)=="21")

t[i].hour=21;

else if(lEvents[i].time.mid(3,2)=="22")

t[i].hour=22;

else

{

if(lEvents[i].time.mid(7,2)=="13")

t[i].hour=13;

if(lEvents[i].time.mid(5,2)=="08")

t[i].hour=8;

if(lEvents[i].time.mid(5,2)=="19")

t[i].hour=19;

}

}

}

冲突检测：

EventInfo info;

info.name = ui->name->text();

info.time = ui->time->text();

info.location = ui->location->text();

//冲突检测

m\_ptreventSql = eventSql::getinstance();

m\_ptreventSql ->init();

auto cnt = m\_ptreventSql ->getEveCnt();

QList<EventInfo> lEvents = m\_ptreventSql->getPageEvent(0,cnt);

bool f = 1;

for(int i = 0;i<lEvents.size()&&f;i++)

{

if(lEvents[i].time.mid(1,4)==info.time.mid(1,4))

f=0;

}

if(f)

{

auto ptr = eventSql::getinstance();

ptr->addEve(info);

QMessageBox::information(nullptr,"个人活动","添加成功");

qDebug()<<"用户添加了个人活动";

}

else

{

QMessageBox::information(nullptr,"警告","时间冲突！");

qDebug()<<"用户添加个人活动时检测到时间冲突";

}

模块二、校园导航

1. **概述**

本导航系统通过如下图的图形界面与用户交互，导航系统是由以课程系统为主界面的副系统，并由主界面进入，实现的交互功能如图所示。



**主界面与导航界面切换按钮**

**切换本部与沙河校区地图按钮**

**虚拟时间显示**

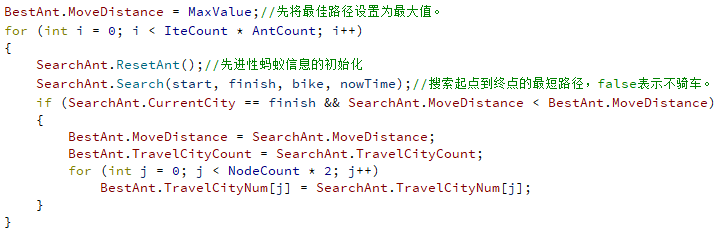
1. **导航算法步骤**

#define IteCount 1000 //迭代次数

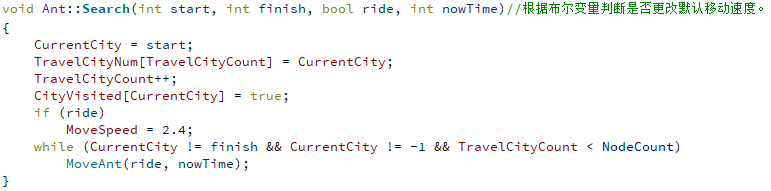
#define AntCount 100 //蚂蚁数量，循环搜索

以最短路径策略为例：

先将路径设置成最大值后，开始进行循环搜索，总共搜寻的次数等于IteCount \* AntCount即为迭代次数乘以蚂蚁数量，SearchAnt为Ant类的变量，用来搜寻路径，每次循环搜索时都先将蚂蚁信息初始化，搜索起点到终点的最短路径，然后if语句判断当前所在结点编号是否为终点并且此次循环的路径距离小于在之前的最佳路径，则将最佳路径替换为现在的路径，如此循环，直到最终找到最佳路径。



1. 以下为Search函数



开始之后，在初始化数据后进入while循环，如果当前结点不为终点，当前结点没有卡死，并且走过结点数量小于总结点数量，则一直向下一个结点移动

1. 以下为MoveAnt函数

先进行下一个结点的选择，如果下一个结点选择为-1则当前结点为-1，结束这次搜索，若不是，则进行四个方向的判断，若下一个结点等于当前结点四个方向中某一个方向的结点标号，则先计算下一结点道路拥挤度下的速度，然后将移动距离加上当前结点到下一结点的距离，随后根据是否跨校区和校园车次时间和等待时间将移动时间进行相应改变，之后向目前走过所以结点数组TravelCityNum最后加入下一结点，并且将下一结点标记为已走过结点，走过结点数量加一

1. 以下为SelectNextCityNum



通过对四个方向的查询，查看所以能到达的方向，再通过随机种子，在能到的所以方向中随机选择一个方向作为下一个前进的结点

最短时间基本原理相同，只是会多考虑校车和公交车的等车时间以及班次，再次不再详细描述。

模块三、模拟系统

**一、模拟系统时间**

**基本思想：**

使用QTimerEvent类定时器事件，用starttimer开启一个定时器，根据模拟时间要求，TimerID = startTimer(1000/360)，即现实时间的1/360秒为虚拟系统的1秒，每1/360秒，实现虚拟系统的时间的计数（秒分时日的加一进位），结果用QPaintEvent打印。

**核心算法：**

实现虚拟系统的时间的计数，秒分时日的加一进位。

构造函数里的初始化：

MainWindwo::**MainWindwo**(QWidget \*parent) :

QWidget(parent),

ui(new Ui::MainWindwo),

m\_ptreventSql(nullptr)

{

ui->setupUi(this);

day = 1;

hour = 0;

minute = 0;

second = 0;

myday = -1;

myhour = -1;

myminute = -1;

TimerID = startTimer(1000/360);

}

成员函数实现计数：

void MainWindwo::***timerEvent***(QTimerEvent \* e)

{

if (e->timerId() == TimerID)

{

second++;

if (second == 60)

{

second = 0;

minute++;//秒向分的进位

if (minute == 60)

{

minute = 0;

hour++;//分向时的进位

if (hour == 24)

{

hour = 0;

day++;//时向日的进位

if (day == 8)

day = 1;//一周过去

}

}

}

}

repaint();

}

1. **闹钟设置**

**基本思想：**

使用QDialog类，弹出对话框让用户选择时间段来设置闹钟。判断虚拟系统的时间是否为设置的时间，是的话用QMessageBox弹出提示消息。

检测用户输入：

void dlg\_setalarm::**on\_Button\_accepted**()

{

setday = ui->weekBox->currentIndex();

sethour = ui->hourBox->currentIndex();

setminute = ui->minuteBox->currentIndex();

}

setday转换为myday用以判断：

switch(mgr->sethour){

case 0:

myhour=10;

break;

case 1:

myhour=14;

break;

case 2:

myhour=18;

break;

}

switch(mgr->setminute){

case 0:

myminute=0;

break;

case 1:

myminute=30;

break;

case 2:

myminute=45;

break;

}

闹钟以QMessageBox形式输出：

//每天的闹钟中午十二点响起

if (hour == 12 && minute == 0 && second == 0)

{

QMessageBox::information(nullptr,"信息","午休时间到！");

}

//每周的闹钟周一早六点响起

if(day == 1 && hour == 6 && minute == 0 && second == 0)

{

QMessageBox::information(nullptr,"信息","又到周一了，快起来上早八！");

}

//建立闹钟

if (day == myday &&hour == myhour && minute == myminute && second == 0)

{

QMessageBox::information(nullptr,"信息","闹钟响了！");

}

优点：实现起来简单，逻辑比较直接。

缺点：只能实现对特定时间的闹钟的设置且闹钟无法精确到秒（默认为00秒）

**三、日志文件**

**基本思想：**

每次用户进行某操作的时候，用qDebug()<<“操作信息”记录下来；

使用 qInstallMsgHandler 函数重定向qDebug、qInfo、qWarning、qCritical、qFatal等宏，输出到txt文件。

**核心算法：**

void **WriteData**(QtMsgType type, const QMessageLogContext &context, const QString &msg)

{

// 加锁

static QMutex mutex;

mutex.lock();

QString text;

switch (type)

{

case QtDebugMsg:

text = QString("Debug:");

break;

case QtWarningMsg:

text = QString("Warning:");

break;

case QtCriticalMsg:

text = QString("Critical:");

break;

case QtFatalMsg:

text = QString("Fatal:");

break;

case QtInfoMsg:

text = QString("Info:");

break;

default:break;

}

// 设置输出信息格式

QString context\_info = QString("File:(%1) Line:(%2) Function:(%3)").arg(QString(context.file)).arg(context.line).arg(context.function);

QString current\_date\_time = QDateTime::currentDateTime().toString("yyyy-MM-dd hh:mm:ss ");

QString current\_date = QString("(%1)").arg(current\_date\_time);

QString message = QString("%1 %2").arg(current\_date).arg(msg);

// 输出信息至文件中（读写、追加形式）

QFile file("log.txt");

file.*open*(QIODevice::WriteOnly | QIODevice::Append);

QTextStream text\_stream(&file);

text\_stream << message << "\r\n";

file.flush();

file.*close*();

// 解锁

mutex.unlock();

}

截图：