**电 子 科 技 大 学**

**综合课程设计报告**

**学生姓名：**关文聪  **学 号：**2016060601008

**指导教师：** 林劼、张栗粽

**实验地点：**电子科技大学 计算机科学与工程学院

**实验时间：**2018年3月-2018年7月

**一、实验室名称：** 国家级计算机实验教学中心

**二、实验项目名称：**

1、最优物流路线计算实验

2、多进程多用户文件一致性读写访问设计实现

3、SQL解析器设计实现

4、互联网+智慧物流质询系统设计实现

**三、实验学时：**6+8+8+10=32

**四、实验原理：**

1、数据结构的线性数据结构、树数据结构、图数据结构的运用。

2、利用操作系统中进程并行，互斥和生产消费者问题实现对文件的数据写入和查询访问。

3、词法、语法分析和语义分析知识；运用语法和语义分析知识实现简单SQL解析器

4、综合实验1,2,3的知识，设计实现整体的软件系统

**五、实验目的：**

1、掌握数据结构的线性数据结构、树数据结构、图数据结构的运用。掌握对数据进行存储和输出的方法。掌握对数据的排序算法以及基于图的存储结构求最短路径的算法。

2、掌握对文件的读取、访问、查询、写入、修改等操作，掌握操作系统中多进程互斥和并行操作的实现方法，解决生产者问题等。

3、掌握编译原理中词法分析、语法分析、语义分析等相关内容，并综合运用实现简单SQL解析器

4、掌握对接整合实验程序的方法，设计实现整体的软件系统并掌握测试、排查错误的方法。

1. **实验内容：**

1、给定物流节点信息，物品信息，节点间代价计算方法要求学生完成以下实验内容。

(1) 根据物品信息综合计算物流物品的优先级别，根据物流优先级别排序物流物品，根据排序结果对物流物品进行逐个发货。

(3) 根据物流物品的物流条件信息，归类物流物品到物流方案类型，物流方案类型可包括：价格最小物流方案，时间最短物流方案、综合最优方案、航空物流方案等。并运用树型结构存储所有的物流物品到划分的物流方案中。

(4) 根据给定的物流节点信息，计算各类物流方案下的物流最短路径

(5) 根据物流最短路径，物流方案和物流优先级发送货物

2、实现多用户多进程环境下文件中数据记录的读写、查询操作。主要实验内容如下。

(1)编程实现文件中记录数据的存储、读写和记录的简单查询与索引查询函数。能够实现单用户和进行对文件数据的写入与查询。

(2)设计实现数据表的文件存储方式，能在文件中存储多张数据表，并写入和查询指定 数据表中的记录。

(2)实现多进程对文件记录的互斥写入与查询访问，保证记录数据的一致性。

(3)基于锁机制保障多用户对文件中记录数据的写如与查询一致性操作。

3、SQL语句的解析的设计。SQL解释器的实质为制定相关文法规则，对用户输入字符串进行语法语义分析，进而采用相应的操作。本实验要求实现部分SQL语句的功能，包括Select语句，Insert语句和Update，创建表语句的解析并对接实验2中对应的创建表、写入和查询函数实现数据表创建、写入和查询操作。

* 1. 构建词法、语法及语义分析程序实现部分SQL语句的解析，包括Select语句， Insert语句和Update语句，创建表语句的解析。
  2. 构建相应的语义子程序。
  3. 将语义子程序对接底层实验2中所实现的各个数据操作函数，实现增删查改等数据库功能。

4、实现智慧物流质询系统，系统具体要求如下。

(1)结合实验3,2构建物流节点信息表，实现物流节点信息的数据库存储

(3) 结合实验3,2，构建物品信息表，实现物品信息的存储

(4) 结合实验3以SQL语句，对物流节点信息进行增删改查

(5) 结合实验3以SQL语句，对物品信息进行增删改查

(6) 结合实验1，实现物品的优先级排序和物流方案分类

(7) 节点信息会动态变化，因此结合实验1，每个物品需要动态计算物流最短路径的实现。

（8）模拟物品的物流状态，用户可以对物件的物流状态进行查询。

**七、实验器材（设备、元器件）：**

PC机

操作系统：Microsoft Windows 10专业版 1803

文本编辑器：Sublime Text 3 Dev Build 3175

编译器：Python 3.6.4、Parser Generator 2.07、Microsoft Visual C++ 6.0

导入Python库：os、numpy、psutil、random

集成开发环境：JetBrains PyCharm（Professional Edition）

**八、数据结构与程序：**

1、最优物流路线计算实验（主要函数）：

# 功能：创建文件，随机生成物品和节点数据信息  
# 在程序运行最开始被调用，如果目录下对应文件不存在就会创建  
def createFile():  
 if (os.path.exists("sql") == False):  
 os.mkdir("sql") # 创建名为sql的文件夹  
 if (os.path.exists(os.getcwd() + "\\sql\\database.dat") == False):  
 with open(os.getcwd() + "\\sql\\database.dat", "w+") as fp:  
 fp.write("test\n") # 创建名为database.dat的数据库记录文件，记录数据库信息test  
 if (os.path.exists(os.getcwd() + "\\sql\\test") == False):  
 os.mkdir(os.getcwd() + "\\sql\\test") # 创建sql文件夹下名为test的文件夹  
 if (os.path.exists(os.getcwd() + "\\sql\\test\\tables.dat") == False):  
 with open(os.getcwd() + "\\sql\\test\\tables.dat", "w+") as fp: # 创建名为tables.dat的表记录文件，记录表信息items和nodes  
 fp.write("nodes\tnode\tINT 4\n")  
 for i in range(7):  
 fp.write("nodes\tdistance" + str(i + 1) + "\t" + "INT 4\n")  
 fp.write("items Id INT 4\n")  
 fp.write("items Price INT 4\n")  
 fp.write("items Fresh INT 4\n")  
 fp.write("items Time INT 4\n")  
 fp.write("items VIP INT 4\n")  
 fp.write("items Weight INT 4\n")  
 fp.write("items Start INT 4\n")  
 fp.write("items End INT 4\n")  
 fp.write("items Status INT 4\n")  
 if (os.path.exists(os.getcwd() + "\\sql\\test\\items.txt") == False):  
 with open(os.getcwd() + "\\sql\\test\\items.txt", "w+") as fp: # 创建items表，随机生成一些数据写入  
 fp.write("9\n")  
 fp.write("Id\tPrice\tFresh\tTime\tVIP\tWeight\tStart\tEnd\tStatus\n")  
 for i in range(10):  
 fp.write(  
 str(i + 1) + "\t" + str(random.randint(1, 100)) + "\t" + str(random.randint(0, 1)) + "\t" + str(  
 random.randint(8, 72)) + "\t" + str(random.randint(0, 1)) + "\t" + str(  
 random.randint(1, 100)) + "\t" + str(  
 random.randint(1, 7)) + "\t" + str(random.randint(1, 7)) + "\t" + str(  
 random.randint(1, 3)) + "\n")  
 if (os.path.exists(os.getcwd() + "\\sql\\test\\nodes.txt") == False):  
 with open(os.getcwd() + "\\sql\\test\\nodes.txt", "w+") as fp: # 创建nodes表，随机生成一些数据写入  
 fp.write("8\n")  
 fp.write("node\tdistance1\tdistance2\tdistance3\tdistance4\tdistance5\tdistance6\tdistance7\n")  
 for i in range(7):  
 fp.write(str(i + 1) + "\t")  
 for j in range(7):  
 fp.write(str(0 if i == j else random.randint(50, 500)) + "\t")  
 fp.write("\n")  
  
  
# 功能:读取nodes表文件，将节点信息用int类型矩阵存储并返回  
def getNodesMatrix():  
 nodesMatrix = numpy.loadtxt(os.getcwd() + "\\sql\\test\\nodes.txt", dtype=int,  
 skiprows=2)  
 return nodesMatrix  
  
  
# 功能:读取nodes表文件，将距离信息用int类型矩阵存储，使矩阵对称化并返回  
def getDistanceMatrix():  
 distanceMatrix = numpy.loadtxt(os.getcwd() + "\\sql\\test\\nodes.txt",  
 dtype=int, skiprows=2, usecols=(1, 2, 3, 4, 5, 6, 7))  
 for i in range(7):  
 for j in range(7):  
 distanceMatrix[i][j] = distanceMatrix[j][i] # 将距离矩阵对称化  
 return distanceMatrix  
  
  
# 功能:读取items表文件，将节点信息用int类型矩阵存储，计算综合权重并返回  
def getItemsMatrix():  
 itemsMatrix = numpy.loadtxt(os.getcwd() + "\\sql\\test\\items.txt", dtype=int,  
 skiprows=2)  
 for i in range(len(itemsMatrix)):  
 itemsMatrix[i][5] = itemsMatrix[i][2] \* 2000 + itemsMatrix[i][4] \* 1000 - itemsMatrix[i][  
 3] \* 5 - itemsMatrix[i][1] \* 2 # 每个物品计算出综合权重  
 return itemsMatrix  
  
  
# 将计算完权重的items表重新写到文件items.txt中  
def updateItemsFile(file, itemsMatrix):  
 with open(file, "w+") as fp:  
 fp.write("9\n")  
 fp.write("Id\tPrice\tFresh\tTime\tVIP\tWeight\tStart\tEnd\tStatus\n")  
 for i in range(len(itemsMatrix)):  
 fp.write(  
 str(itemsMatrix[i][0]) + "\t" + str(itemsMatrix[i][1]) + "\t" + str(  
 itemsMatrix[i][2]) + "\t" + str(  
 itemsMatrix[i][3]) + "\t" + str(itemsMatrix[i][4]) + "\t" + str(  
 itemsMatrix[i][5]) + "\t" + str(  
 itemsMatrix[i][6]) + "\t" + str(itemsMatrix[i][7]) + "\t" + str(  
 itemsMatrix[i][8]) + "\n")  
  
  
# 将距离节点信息对称化的nodes表重新写到文件nodes.txt中  
def updateNodesMatrixFile(file, distanceMatrix):  
 with open(file, "w+") as fp:  
 fp.write("8\n")  
 fp.write("node\tdistance1\tdistance2\tdistance3\tdistance4\tdistance5\tdistance6\tdistance7\n")  
 for i in range(7):  
 fp.write(str(i + 1) + "\t")  
 for j in range(7):  
 fp.write(str(distanceMatrix[i][j]) + "\t")  
 fp.write("\n")  
  
  
# 按时间从小到大排序并输出  
def timeSort(itemsMatrix):  
 timeArray = numpy.loadtxt(os.getcwd() + "\\sql\\test\\items.txt", dtype=int,  
 skiprows=2, usecols=3)  
 timeArray = sorted(timeArray) # 按时间从小到大排序  
 for j in range(len(timeArray)): # 匹配对应物品并输出  
 for i in range(len(itemsMatrix)):  
 if (itemsMatrix[i][3] == timeArray[j]):  
 print("ID：" + str(itemsMatrix[i][0]) + "\t" + "Price：" + str(  
 itemsMatrix[i][1]) + "\t" + "Fresh：" + str(  
 itemsMatrix[i][2]) + "\t" + "Time：" + str(itemsMatrix[i][3]) + "\t" + "VIP：" + str(  
 itemsMatrix[i][4]) + "\t" + "Weight：" + str(  
 itemsMatrix[i][5]) + "\t" + "Start：" + str(  
 itemsMatrix[i][6]) + "\t" + "End：" + str(  
 itemsMatrix[i][7]) + "\t" + "Status：" + str(  
 itemsMatrix[i][8]))  
 showPath(path, itemsMatrix[i][6], itemsMatrix[i][7])  
 if (itemsMatrix[i][8] == 1):  
 print("状态：未发送")  
 elif (itemsMatrix[i][8] == 2):  
 print("状态：发送中")  
 elif (itemsMatrix[i][8] == 3):  
 print("状态：已送达")  
  
  
# 按费用从低到高排序并输出  
def priceSort(itemsMatrix):  
 priceArray = numpy.loadtxt(os.getcwd() + "\\sql\\test\\items.txt", dtype=int,  
 skiprows=2, usecols=1)  
 priceArray = sorted(priceArray) # 按费用从低到高排序  
 for j in range(len(priceArray)): # 匹配对应物品并输出  
 for i in range(len(itemsMatrix)):  
 if (itemsMatrix[i][1] == priceArray[j]):  
 print("ID：" + str(itemsMatrix[i][0]) + "\t" + "Price：" + str(  
 itemsMatrix[i][1]) + "\t" + "Fresh：" + str(  
 itemsMatrix[i][2]) + "\t" + "Time：" + str(itemsMatrix[i][3]) + "\t" + "VIP：" + str(  
 itemsMatrix[i][4]) + "\t" + "Weight：" + str(  
 itemsMatrix[i][5]) + "\t" + "Start：" + str(  
 itemsMatrix[i][6]) + "\t" + "End：" + str(  
 itemsMatrix[i][7]) + "\t" + "Status：" + str(  
 itemsMatrix[i][8]))  
 showPath(path, itemsMatrix[i][6], itemsMatrix[i][7])  
 if (itemsMatrix[i][8] == 1):  
 print("状态：未发送")  
 elif (itemsMatrix[i][8] == 2):  
 print("状态：发送中")  
 elif (itemsMatrix[i][8] == 3):  
 print("状态：已送达")  
  
  
# 按综合权重从高到低排序并输出  
def weightSort(itemsMatrix):  
 weightArray = numpy.loadtxt(os.getcwd() + "\\sql\\test\\items.txt", dtype=int,  
 skiprows=2, usecols=5)  
 weightArray = sorted(weightArray, reverse=True) # 按综合权重从高到低排序  
 for j in range(len(weightArray)): # 匹配对应物品并输出  
 for i in range(len(itemsMatrix)):  
 if (itemsMatrix[i][5] == weightArray[j]):  
 print("ID：" + str(itemsMatrix[i][0]) + "\t" + "Price：" + str(  
 itemsMatrix[i][1]) + "\t" + "Fresh：" + str(  
 itemsMatrix[i][2]) + "\t" + "Time：" + str(itemsMatrix[i][3]) + "\t" + "VIP：" + str(  
 itemsMatrix[i][4]) + "\t" + "Weight：" + str(  
 itemsMatrix[i][5]) + "\t" + "Start：" + str(  
 itemsMatrix[i][6]) + "\t" + "End：" + str(  
 itemsMatrix[i][7]) + "\t" + "Status：" + str(  
 itemsMatrix[i][8]))  
 showPath(path, itemsMatrix[i][6], itemsMatrix[i][7])  
 if (itemsMatrix[i][8] == 1):  
 print("状态：未发送")  
 elif (itemsMatrix[i][8] == 2):  
 print("状态：发送中")  
 elif (itemsMatrix[i][8] == 3):  
 print("状态：已送达")  
  
  
# 弗洛伊德算法：计算全源最短路径并用矩阵形式存储  
def floyd(distanceMatrix):  
 temp = numpy.zeros((7, 7)) # 7\*7的矩阵用于临时存储和计算  
 path = numpy.zeros((7, 7)) # 设置了7个节点  
  
 # 弗洛伊德算法  
 for i in range(7):  
 for j in range(7):  
 temp[i][j] = distanceMatrix[i][j]  
 if (i != j and distanceMatrix[i][j] < 1000):  
 path[i][j] = i  
 else:  
 path[i][j] = -1  
  
 for k in range(7):  
 for i in range(7):  
 for j in range(7):  
 if (temp[i][j] > temp[i][k] + temp[k][j]):  
 temp[i][j] = temp[i][k] + temp[k][j]  
 path[i][j] = path[k][j]  
 return path  
  
  
# 输出最短路径节点信息  
def showPath(path, from\_s, to):  
 if (from\_s == 1):  
 from\_ = 1  
 elif (from\_s == 2):  
 from\_ = 2  
 elif (from\_s == 3):  
 from\_ = 3  
 elif (from\_s == 4):  
 from\_ = 4  
 elif (from\_s == 5):  
 from\_ = 5  
 elif (from\_s == 6):  
 from\_ = 6  
 else:  
 from\_ = 7  
 if (to == 1):  
 to\_ = 1  
 elif (to == 2):  
 to\_ = 2  
 elif (to == 3):  
 to\_ = 3  
 elif (to == 4):  
 to\_ = 4  
 elif (to == 5):  
 to\_ = 5  
 elif (to == 6):  
 to\_ = 6  
 else:  
 to\_ = 7  
 if (to\_ > 7 or to\_ < 1 or from\_ > 7 or from\_ < 1):  
 print("您输入的节点不存在！")  
 t = from\_ - 1  
 f = to\_ - 1  
 if (t == f):  
 print("出发点不能与终点相同")  
 return  
 print("距离最短的路径为：", end="")  
 print(from\_, end=" ")  
 print("--->", end="")  
 path.reshape(7, 7).dtype = int  
 while (path[f][t] != f):  
 print(chr(int(path[f][t] + ord('1'))), end="")  
 print("--->", end="")  
 t = int(path[f][t])  
  
 print(to\_)

2、多进程多用户文件一致性读写访问设计实现（主要函数）：

# 功能：检查进程，检测系统中是否有sql.exe进程正在运行  
# 若有sql.exe进程，返回True，若没有，返回False  
def checkProcess():  
 flag = False  
 for proc in psutil.process\_iter(attrs=['pid', 'name']):  
 if (proc.name() == "sql.exe"):  
 flag = True  
 return flag

if \_\_name\_\_ == '\_\_main\_\_':  
 createFile() # 如果对应文件不存在先创建  
 while (True):  
 flag = checkProcess()  
 if (flag == False):  
 choice = input("\n请选择功能：1：查看所有物品 2：查看所有节点 3：航空物流方案 4：时间排序 5：价格排序 6：综合权重排序（降序） 7：SQL输入 8：退出系统\n")  
 itemsMatrix = getItemsMatrix() # 得到物品矩阵  
 distanceMatrix = getDistanceMatrix() # 得到距离矩阵  
 path = floyd(distanceMatrix) # 调用弗洛伊德算法求全源最短路径  
 updateItemsFile(os.getcwd() + "\\sql\\test\\items.txt", itemsMatrix) # 将计算完权重的items表重新写入文件  
 updateNodesMatrixFile(os.getcwd() + "\\sql\\test\\nodes.txt", distanceMatrix) # 将对称化的nodes表重新写入文件  
 nodeMatrix = getNodesMatrix() # 得到节点信息矩阵  
 distanceMatrix = getDistanceMatrix() # 得到距离矩阵  
 itemsMatrix = getItemsMatrix() # 得到物品矩阵  
 if (int(choice) == 1):  
 flag = checkProcess()  
 if (flag == False):  
 for i in range(len(itemsMatrix)):  
 print("ID：" + str(itemsMatrix[i][0]) + "\t" + "Price：" + str(  
 itemsMatrix[i][1]) + "\t" + "Fresh：" + str(  
 itemsMatrix[i][2]) + "\t" + "Time：" + str(itemsMatrix[i][3]) + "\t" + "VIP：" + str(  
 itemsMatrix[i][4]) + "\t" + "Weight：" + str(  
 itemsMatrix[i][5]) + "\t" + "Start：" + str(  
 itemsMatrix[i][6]) + "\t" + "End：" + str(  
 itemsMatrix[i][7]) + "\t" + "Status：" + str(  
 itemsMatrix[i][8]))  
 showPath(path, itemsMatrix[i][6], itemsMatrix[i][7])  
 if (itemsMatrix[i][8] == 1):  
 print("状态：未发送")  
 elif (itemsMatrix[i][8] == 2):  
 print("状态：发送中")  
 elif (itemsMatrix[i][8] == 3):  
 print("状态：已送达")  
 else:  
 print("检测到有sql.exe进程正在操作，请等待！")  
 continue  
 elif (int(choice) == 2):  
 flag = checkProcess()  
 if (flag == False):  
 for i in range(len(nodeMatrix)):  
 print("Node：" + str(nodeMatrix[i][0]) + "\t" + "Distance1：" + str(  
 nodeMatrix[i][1]) + "\t" + "Distance2：" + str(  
 nodeMatrix[i][2]) + "\t" + "Distance3：" + str(  
 nodeMatrix[i][3]) + "\t" + "Distance4：" + str(  
 nodeMatrix[i][4]) + "\t" + "Distance5：" + str(  
 nodeMatrix[i][5]) + "\t" + "Distance6：" + str(  
 nodeMatrix[i][6]) + "\t" + "Distance7：" + str(  
 nodeMatrix[i][7]) + "\n")  
 else:  
 print("检测到有sql.exe进程正在操作，请等待！")  
 continue  
 elif (int(choice) == 3):  
 flag = checkProcess()  
 if (flag == False):  
 print("对生鲜物品要采用航空运输：")  
 for i in range(len(itemsMatrix)):  
 if (int(itemsMatrix[i][2]) == 1):  
 print("ID：" + str(itemsMatrix[i][0]) + "\t" + "Price：" + str(  
 itemsMatrix[i][1]) + "\t" + "Fresh：" + str(  
 itemsMatrix[i][2]) + "\t" + "Time：" + str(  
 itemsMatrix[i][3]) + "\t" + "VIP：" + str(  
 itemsMatrix[i][4]) + "\t" + "Weight：" + str(  
 itemsMatrix[i][5]) + "\t" + "Start：" + str(  
 itemsMatrix[i][6]) + "\t" + "End：" + str(itemsMatrix[i][7]))  
 print("航空直达：" + str(itemsMatrix[i][6]) + "→" + str(itemsMatrix[i][7]))  
 else:  
 print("检测到有sql.exe进程正在操作，请等待！")  
 continue  
 elif (int(choice) == 4):  
 flag = checkProcess()  
 if (flag == False):  
 print("按时间排序：\n")  
 timeSort(itemsMatrix)  
 else:  
 print("检测到有sql.exe进程正在操作，请等待！")  
 continue  
 elif (int(choice) == 5):  
 flag = checkProcess()  
 if (flag == False):  
 print("按价格排序：\n")  
 priceSort(itemsMatrix)  
 else:  
 print("检测到有sql.exe进程正在操作，请等待！")  
 continue  
 elif (int(choice) == 6):  
 flag = checkProcess()  
 if (flag == False):  
 print("按综合权重降序排序：")  
 weightSort(itemsMatrix)  
 else:  
 print("检测到有sql.exe进程正在操作，请等待！")  
 continue  
 elif (int(choice) == 7):  
 flag = checkProcess()  
 if (flag == False):  
 os.system(os.getcwd() + "\\sql.exe")  
 else:  
 print("检测到有sql.exe进程正在操作，请等待！")  
 continue  
 elif (int(choice) == 8):  
 print("欢迎下次使用！再见！")  
 break  
 else:  
 continue  
 else:  
 print("检测到有sql.exe进程正在操作，请等待！")

3、SQL解析器设计实现：

（Lex进行词法分析，Yacc进行语法分析。sqlparser.l文件是Lex词法分析文件，sqlparser.y文件是Yacc语法分析文件。二者通过Parser Generator编译生成mylexer.c、mylexer.h、myparser.c、myparser.h文件，再添加到Microsoft Visual C++ 6.0的空工程中，编译生成SQL解释器sql.exe，部分文件代码较长，请参照压缩包）

Sqlparser.l文件：

%{

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

mylexer.l

ParserWizard generated Lex file.

Date: 2018年7月10日

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include "myparser.h"

#include<stdlib.h>

#include<stdio.h>

#include<string.h>

%}

/////////////////////////////////////////////////////////////////////////////

// declarations section

//实现大小写不敏感

CREATE [Cc][Rr][Ee][Aa][Tt][Ee]

USE [Uu][Ss][Ee]

SHOW [Ss][Hh][Oo][Ww]

INSERT [Ii][Nn][Ss][Ee][Rr][Tt]

SELECT [Ss][Ee][Ll][Ee][Cc][Tt]

UPDATE [Uu][Pp][Dd][Aa][Tt][Ee]

SET [Ss][Ee][Tt]

DELETE [Dd][Ee][Ll][Ee][Tt][Ee]

DROP [Dd][Rr][Oo][Pp]

AND [Aa][Nn][Dd]

WHERE [Ww][Hh][Ee][Rr][Ee]

OR [Oo][Rr]

FROM [Ff][Rr][Oo][Mm]

INTO [Ii][Nn][Tt][Oo]

VALUES [Vv][Aa][Ll][Uu][Ee][Ss]

EXIT [Ee][Xx][Ii][Tt]

DATABASE [Dd][Aa][Tt][Aa][Bb][Aa][Ss][Ee]

DATABASES [Dd][Aa][Tt][Aa][Bb][Aa][Ss][Ee][Ss]

TABLE [Tt][Aa][Bb][Ll][Ee]

TABLES [Tt][Aa][Bb][Ll][Ee][Ss]

CHAR [Cc][Hh][Aa][Rr]

INT [Ii][Nn][Tt]

ID [A-Za-z][A-Za-z0-9\_]\*

digit [0-9]

digits {digit}+

optional\_fraction ("."{digits})?

optional\_exponent (E[+-]?{digits})?

// place any declarations here

%%

/////////////////////////////////////////////////////////////////////////////

// rules section

[ ]+ ;//过滤空格

//关键字表，识别SQL语句对应的关键字

{CREATE} {return CREATE;}

{USE} {return USE;}

{SHOW} {return SHOW;}

{INSERT} {return INSERT;}

{SELECT} {return SELECT;}

{UPDATE} {return UPDATE;}

{SET} {return SET;}

{DELETE} {return DELETE;}

{DROP} {return DROP;}

{INTO} {return INTO;}

{VALUES} {return VALUES;}

{FROM} {return FROM;}

{WHERE} {return WHERE;}

{AND} {return AND;}

{OR} {return OR;}

{EXIT} {return EXIT;}

{TABLE} {return TABLE;}

{TABLES} {return TABLES;}

{DATABASE} {return DATABASE;}

{DATABASES} {return DATABASES;}

{CHAR} {return CHAR;}

{INT} {return INT;}

{ID} {

yylval.yych=(char \*)malloc(strlen(yytext)+1);

strcpy(yylval.yych, yytext);

return ID;

}

{digits}{optional\_fraction}{optional\_exponent} {

yylval.yych=(char \*)malloc(strlen(yytext)+1);

strcpy(yylval.yych, yytext);

return NUMBER;

}

//符号表，识别SQL语句对应的符号

";" {return ';';}

"(" {return '(';}

")" {return ')';}

"," {return ',';}

"." {return '.';}

"!" {return '!';}

"=" {return '=';}

"<" {return '<';}

">" {return '>';}

"'" {return QUOTE;}

"+" {return '+';}

"-" {return '-';}

"\*" {return '\*';}

"/" {return '/';}

"\n" {return 0;}

// place your Lex rules here

%%

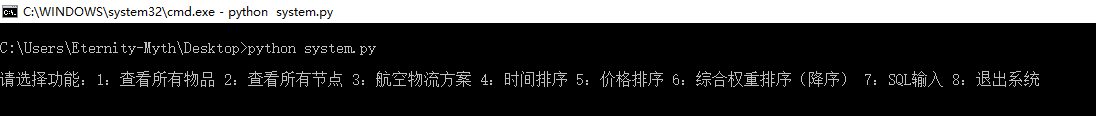
Sqlparser.y文件代码较长，请参照压缩包查看

4、互联网+智慧物流质询系统设计实现（整合以上各模块代码为一个完整系统，注意要导入Python库os、numpy、psutil、random）

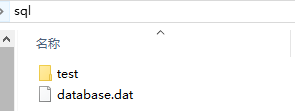
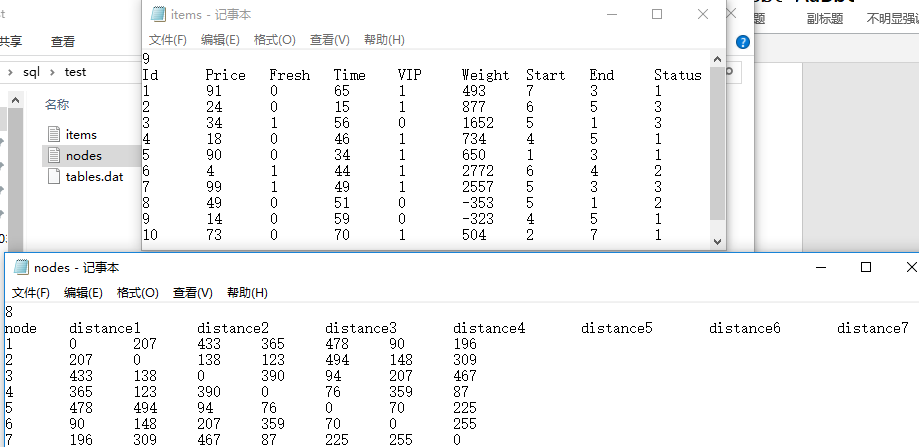
import os  
import numpy  
import psutil  
import random  
  
  
# 功能：创建文件，随机生成物品和节点数据信息  
# 在程序运行最开始被调用，如果目录下对应文件不存在就会创建  
def createFile():  
 if (os.path.exists("sql") == False):  
 os.mkdir("sql") # 创建名为sql的文件夹  
 if (os.path.exists(os.getcwd() + "\\sql\\database.dat") == False):  
 with open(os.getcwd() + "\\sql\\database.dat", "w+") as fp:  
 fp.write("test\n") # 创建名为database.dat的数据库记录文件，记录数据库信息test  
 if (os.path.exists(os.getcwd() + "\\sql\\test") == False):  
 os.mkdir(os.getcwd() + "\\sql\\test") # 创建sql文件夹下名为test的文件夹  
 if (os.path.exists(os.getcwd() + "\\sql\\test\\tables.dat") == False):  
 with open(os.getcwd() + "\\sql\\test\\tables.dat", "w+") as fp: # 创建名为tables.dat的表记录文件，记录表信息items和nodes  
 fp.write("nodes\tnode\tINT 4\n")  
 for i in range(7):  
 fp.write("nodes\tdistance" + str(i + 1) + "\t" + "INT 4\n")  
 fp.write("items Id INT 4\n")  
 fp.write("items Price INT 4\n")  
 fp.write("items Fresh INT 4\n")  
 fp.write("items Time INT 4\n")  
 fp.write("items VIP INT 4\n")  
 fp.write("items Weight INT 4\n")  
 fp.write("items Start INT 4\n")  
 fp.write("items End INT 4\n")  
 fp.write("items Status INT 4\n")  
 if (os.path.exists(os.getcwd() + "\\sql\\test\\items.txt") == False):  
 with open(os.getcwd() + "\\sql\\test\\items.txt", "w+") as fp: # 创建items表，随机生成一些数据写入  
 fp.write("9\n")  
 fp.write("Id\tPrice\tFresh\tTime\tVIP\tWeight\tStart\tEnd\tStatus\n")  
 for i in range(10):  
 fp.write(  
 str(i + 1) + "\t" + str(random.randint(1, 100)) + "\t" + str(random.randint(0, 1)) + "\t" + str(  
 random.randint(8, 72)) + "\t" + str(random.randint(0, 1)) + "\t" + str(  
 random.randint(1, 100)) + "\t" + str(  
 random.randint(1, 7)) + "\t" + str(random.randint(1, 7)) + "\t" + str(  
 random.randint(1, 3)) + "\n")  
 if (os.path.exists(os.getcwd() + "\\sql\\test\\nodes.txt") == False):  
 with open(os.getcwd() + "\\sql\\test\\nodes.txt", "w+") as fp: # 创建nodes表，随机生成一些数据写入  
 fp.write("8\n")  
 fp.write("node\tdistance1\tdistance2\tdistance3\tdistance4\tdistance5\tdistance6\tdistance7\n")  
 for i in range(7):  
 fp.write(str(i + 1) + "\t")  
 for j in range(7):  
 fp.write(str(0 if i == j else random.randint(50, 500)) + "\t")  
 fp.write("\n")  
  
  
# 功能:读取nodes表文件，将节点信息用int类型矩阵存储并返回  
def getNodesMatrix():  
 nodesMatrix = numpy.loadtxt(os.getcwd() + "\\sql\\test\\nodes.txt", dtype=int,  
 skiprows=2)  
 return nodesMatrix  
  
  
# 功能:读取nodes表文件，将距离信息用int类型矩阵存储，使矩阵对称化并返回  
def getDistanceMatrix():  
 distanceMatrix = numpy.loadtxt(os.getcwd() + "\\sql\\test\\nodes.txt",  
 dtype=int, skiprows=2, usecols=(1, 2, 3, 4, 5, 6, 7))  
 for i in range(7):  
 for j in range(7):  
 distanceMatrix[i][j] = distanceMatrix[j][i] # 将距离矩阵对称化  
 return distanceMatrix  
  
  
# 功能:读取items表文件，将节点信息用int类型矩阵存储，计算综合权重并返回  
def getItemsMatrix():  
 itemsMatrix = numpy.loadtxt(os.getcwd() + "\\sql\\test\\items.txt", dtype=int,  
 skiprows=2)  
 for i in range(len(itemsMatrix)):  
 itemsMatrix[i][5] = itemsMatrix[i][2] \* 2000 + itemsMatrix[i][4] \* 1000 - itemsMatrix[i][  
 3] \* 5 - itemsMatrix[i][1] \* 2 # 每个物品计算出综合权重  
 return itemsMatrix  
  
  
# 将计算完权重的items表重新写到文件items.txt中  
def updateItemsFile(file, itemsMatrix):  
 with open(file, "w+") as fp:  
 fp.write("9\n")  
 fp.write("Id\tPrice\tFresh\tTime\tVIP\tWeight\tStart\tEnd\tStatus\n")  
 for i in range(len(itemsMatrix)):  
 fp.write(  
 str(itemsMatrix[i][0]) + "\t" + str(itemsMatrix[i][1]) + "\t" + str(  
 itemsMatrix[i][2]) + "\t" + str(  
 itemsMatrix[i][3]) + "\t" + str(itemsMatrix[i][4]) + "\t" + str(  
 itemsMatrix[i][5]) + "\t" + str(  
 itemsMatrix[i][6]) + "\t" + str(itemsMatrix[i][7]) + "\t" + str(  
 itemsMatrix[i][8]) + "\n")  
  
  
# 将距离节点信息对称化的nodes表重新写到文件nodes.txt中  
def updateNodesMatrixFile(file, distanceMatrix):  
 with open(file, "w+") as fp:  
 fp.write("8\n")  
 fp.write("node\tdistance1\tdistance2\tdistance3\tdistance4\tdistance5\tdistance6\tdistance7\n")  
 for i in range(7):  
 fp.write(str(i + 1) + "\t")  
 for j in range(7):  
 fp.write(str(distanceMatrix[i][j]) + "\t")  
 fp.write("\n")  
  
  
# 按时间从小到大排序并输出  
def timeSort(itemsMatrix):  
 timeArray = numpy.loadtxt(os.getcwd() + "\\sql\\test\\items.txt", dtype=int,  
 skiprows=2, usecols=3)  
 timeArray = sorted(timeArray) # 按时间从小到大排序  
 for j in range(len(timeArray)): # 匹配对应物品并输出  
 for i in range(len(itemsMatrix)):  
 if (itemsMatrix[i][3] == timeArray[j]):  
 print("ID：" + str(itemsMatrix[i][0]) + "\t" + "Price：" + str(  
 itemsMatrix[i][1]) + "\t" + "Fresh：" + str(  
 itemsMatrix[i][2]) + "\t" + "Time：" + str(itemsMatrix[i][3]) + "\t" + "VIP：" + str(  
 itemsMatrix[i][4]) + "\t" + "Weight：" + str(  
 itemsMatrix[i][5]) + "\t" + "Start：" + str(  
 itemsMatrix[i][6]) + "\t" + "End：" + str(  
 itemsMatrix[i][7]) + "\t" + "Status：" + str(  
 itemsMatrix[i][8]))  
 showPath(path, itemsMatrix[i][6], itemsMatrix[i][7])  
 if (itemsMatrix[i][8] == 1):  
 print("状态：未发送")  
 elif (itemsMatrix[i][8] == 2):  
 print("状态：发送中")  
 elif (itemsMatrix[i][8] == 3):  
 print("状态：已送达")  
  
  
# 按费用从低到高排序并输出  
def priceSort(itemsMatrix):  
 priceArray = numpy.loadtxt(os.getcwd() + "\\sql\\test\\items.txt", dtype=int,  
 skiprows=2, usecols=1)  
 priceArray = sorted(priceArray) # 按费用从低到高排序  
 for j in range(len(priceArray)): # 匹配对应物品并输出  
 for i in range(len(itemsMatrix)):  
 if (itemsMatrix[i][1] == priceArray[j]):  
 print("ID：" + str(itemsMatrix[i][0]) + "\t" + "Price：" + str(  
 itemsMatrix[i][1]) + "\t" + "Fresh：" + str(  
 itemsMatrix[i][2]) + "\t" + "Time：" + str(itemsMatrix[i][3]) + "\t" + "VIP：" + str(  
 itemsMatrix[i][4]) + "\t" + "Weight：" + str(  
 itemsMatrix[i][5]) + "\t" + "Start：" + str(  
 itemsMatrix[i][6]) + "\t" + "End：" + str(  
 itemsMatrix[i][7]) + "\t" + "Status：" + str(  
 itemsMatrix[i][8]))  
 showPath(path, itemsMatrix[i][6], itemsMatrix[i][7])  
 if (itemsMatrix[i][8] == 1):  
 print("状态：未发送")  
 elif (itemsMatrix[i][8] == 2):  
 print("状态：发送中")  
 elif (itemsMatrix[i][8] == 3):  
 print("状态：已送达")  
  
  
# 按综合权重从高到低排序并输出  
def weightSort(itemsMatrix):  
 weightArray = numpy.loadtxt(os.getcwd() + "\\sql\\test\\items.txt", dtype=int,  
 skiprows=2, usecols=5)  
 weightArray = sorted(weightArray, reverse=True) # 按综合权重从高到低排序  
 for j in range(len(weightArray)): # 匹配对应物品并输出  
 for i in range(len(itemsMatrix)):  
 if (itemsMatrix[i][5] == weightArray[j]):  
 print("ID：" + str(itemsMatrix[i][0]) + "\t" + "Price：" + str(  
 itemsMatrix[i][1]) + "\t" + "Fresh：" + str(  
 itemsMatrix[i][2]) + "\t" + "Time：" + str(itemsMatrix[i][3]) + "\t" + "VIP：" + str(  
 itemsMatrix[i][4]) + "\t" + "Weight：" + str(  
 itemsMatrix[i][5]) + "\t" + "Start：" + str(  
 itemsMatrix[i][6]) + "\t" + "End：" + str(  
 itemsMatrix[i][7]) + "\t" + "Status：" + str(  
 itemsMatrix[i][8]))  
 showPath(path, itemsMatrix[i][6], itemsMatrix[i][7])  
 if (itemsMatrix[i][8] == 1):  
 print("状态：未发送")  
 elif (itemsMatrix[i][8] == 2):  
 print("状态：发送中")  
 elif (itemsMatrix[i][8] == 3):  
 print("状态：已送达")  
  
  
# 弗洛伊德算法：计算全源最短路径并用矩阵形式存储  
def floyd(distanceMatrix):  
 temp = numpy.zeros((7, 7)) # 7\*7的矩阵用于临时存储和计算  
 path = numpy.zeros((7, 7)) # 设置了7个节点  
  
 # 弗洛伊德算法  
 for i in range(7):  
 for j in range(7):  
 temp[i][j] = distanceMatrix[i][j]  
 if (i != j and distanceMatrix[i][j] < 1000):  
 path[i][j] = i  
 else:  
 path[i][j] = -1  
  
 for k in range(7):  
 for i in range(7):  
 for j in range(7):  
 if (temp[i][j] > temp[i][k] + temp[k][j]):  
 temp[i][j] = temp[i][k] + temp[k][j]  
 path[i][j] = path[k][j]  
 return path  
  
  
# 输出最短路径节点信息  
def showPath(path, from\_s, to):  
 if (from\_s == 1):  
 from\_ = 1  
 elif (from\_s == 2):  
 from\_ = 2  
 elif (from\_s == 3):  
 from\_ = 3  
 elif (from\_s == 4):  
 from\_ = 4  
 elif (from\_s == 5):  
 from\_ = 5  
 elif (from\_s == 6):  
 from\_ = 6  
 else:  
 from\_ = 7  
 if (to == 1):  
 to\_ = 1  
 elif (to == 2):  
 to\_ = 2  
 elif (to == 3):  
 to\_ = 3  
 elif (to == 4):  
 to\_ = 4  
 elif (to == 5):  
 to\_ = 5  
 elif (to == 6):  
 to\_ = 6  
 else:  
 to\_ = 7  
 if (to\_ > 7 or to\_ < 1 or from\_ > 7 or from\_ < 1):  
 print("您输入的节点不存在！")  
 t = from\_ - 1  
 f = to\_ - 1  
 if (t == f):  
 print("出发点不能与终点相同")  
 return  
 print("距离最短的路径为：", end="")  
 print(from\_, end=" ")  
 print("--->", end="")  
 path.reshape(7, 7).dtype = int  
 while (path[f][t] != f):  
 print(chr(int(path[f][t] + ord('1'))), end="")  
 print("--->", end="")  
 t = int(path[f][t])  
  
 print(to\_)  
  
  
# 功能：检查进程，检测系统中是否有sql.exe进程正在运行  
# 若有sql.exe进程，返回True，若没有，返回False  
def checkProcess():  
 flag = False  
 for proc in psutil.process\_iter(attrs=['pid', 'name']):  
 if (proc.name() == "sql.exe"):  
 flag = True  
 return flag  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 createFile() # 如果对应文件不存在先创建  
 while (True):  
 flag = checkProcess()  
 if (flag == False):  
 choice = input("\n请选择功能：1：查看所有物品 2：查看所有节点 3：航空物流方案 4：时间排序 5：价格排序 6：综合权重排序（降序） 7：SQL输入 8：退出系统\n")  
 itemsMatrix = getItemsMatrix() # 得到物品矩阵  
 distanceMatrix = getDistanceMatrix() # 得到距离矩阵  
 path = floyd(distanceMatrix) # 调用弗洛伊德算法求全源最短路径  
 updateItemsFile(os.getcwd() + "\\sql\\test\\items.txt", itemsMatrix) # 将计算完权重的items表重新写入文件  
 updateNodesMatrixFile(os.getcwd() + "\\sql\\test\\nodes.txt", distanceMatrix) # 将对称化的nodes表重新写入文件  
 nodeMatrix = getNodesMatrix() # 得到节点信息矩阵  
 distanceMatrix = getDistanceMatrix() # 得到距离矩阵  
 itemsMatrix = getItemsMatrix() # 得到物品矩阵  
 if (int(choice) == 1):  
 flag = checkProcess()  
 if (flag == False):  
 for i in range(len(itemsMatrix)):  
 print("ID：" + str(itemsMatrix[i][0]) + "\t" + "Price：" + str(  
 itemsMatrix[i][1]) + "\t" + "Fresh：" + str(  
 itemsMatrix[i][2]) + "\t" + "Time：" + str(itemsMatrix[i][3]) + "\t" + "VIP：" + str(  
 itemsMatrix[i][4]) + "\t" + "Weight：" + str(  
 itemsMatrix[i][5]) + "\t" + "Start：" + str(  
 itemsMatrix[i][6]) + "\t" + "End：" + str(  
 itemsMatrix[i][7]) + "\t" + "Status：" + str(  
 itemsMatrix[i][8]))  
 showPath(path, itemsMatrix[i][6], itemsMatrix[i][7])  
 if (itemsMatrix[i][8] == 1):  
 print("状态：未发送")  
 elif (itemsMatrix[i][8] == 2):  
 print("状态：发送中")  
 elif (itemsMatrix[i][8] == 3):  
 print("状态：已送达")  
 else:  
 print("检测到有sql.exe进程正在操作，请等待！")  
 continue  
 elif (int(choice) == 2):  
 flag = checkProcess()  
 if (flag == False):  
 for i in range(len(nodeMatrix)):  
 print("Node：" + str(nodeMatrix[i][0]) + "\t" + "Distance1：" + str(  
 nodeMatrix[i][1]) + "\t" + "Distance2：" + str(  
 nodeMatrix[i][2]) + "\t" + "Distance3：" + str(  
 nodeMatrix[i][3]) + "\t" + "Distance4：" + str(  
 nodeMatrix[i][4]) + "\t" + "Distance5：" + str(  
 nodeMatrix[i][5]) + "\t" + "Distance6：" + str(  
 nodeMatrix[i][6]) + "\t" + "Distance7：" + str(  
 nodeMatrix[i][7]) + "\n")  
 else:  
 print("检测到有sql.exe进程正在操作，请等待！")  
 continue  
 elif (int(choice) == 3):  
 flag = checkProcess()  
 if (flag == False):  
 print("对生鲜物品要采用航空运输：")  
 for i in range(len(itemsMatrix)):  
 if (int(itemsMatrix[i][2]) == 1):  
 print("ID：" + str(itemsMatrix[i][0]) + "\t" + "Price：" + str(  
 itemsMatrix[i][1]) + "\t" + "Fresh：" + str(  
 itemsMatrix[i][2]) + "\t" + "Time：" + str(  
 itemsMatrix[i][3]) + "\t" + "VIP：" + str(  
 itemsMatrix[i][4]) + "\t" + "Weight：" + str(  
 itemsMatrix[i][5]) + "\t" + "Start：" + str(  
 itemsMatrix[i][6]) + "\t" + "End：" + str(itemsMatrix[i][7]))  
 print("航空直达：" + str(itemsMatrix[i][6]) + "→" + str(itemsMatrix[i][7]))  
 else:  
 print("检测到有sql.exe进程正在操作，请等待！")  
 continue  
 elif (int(choice) == 4):  
 flag = checkProcess()  
 if (flag == False):  
 print("按时间排序：\n")  
 timeSort(itemsMatrix)  
 else:  
 print("检测到有sql.exe进程正在操作，请等待！")  
 continue  
 elif (int(choice) == 5):  
 flag = checkProcess()  
 if (flag == False):  
 print("按价格排序：\n")  
 priceSort(itemsMatrix)  
 else:  
 print("检测到有sql.exe进程正在操作，请等待！")  
 continue  
 elif (int(choice) == 6):  
 flag = checkProcess()  
 if (flag == False):  
 print("按综合权重降序排序：")  
 weightSort(itemsMatrix)  
 else:  
 print("检测到有sql.exe进程正在操作，请等待！")  
 continue  
 elif (int(choice) == 7):  
 flag = checkProcess()  
 if (flag == False):  
 os.system(os.getcwd() + "\\sql.exe")  
 else:  
 print("检测到有sql.exe进程正在操作，请等待！")  
 continue  
 elif (int(choice) == 8):  
 print("欢迎下次使用！再见！")  
 break  
 else:  
 continue  
 else:  
 print("检测到有sql.exe进程正在操作，请等待！")

程序运行结果展示：

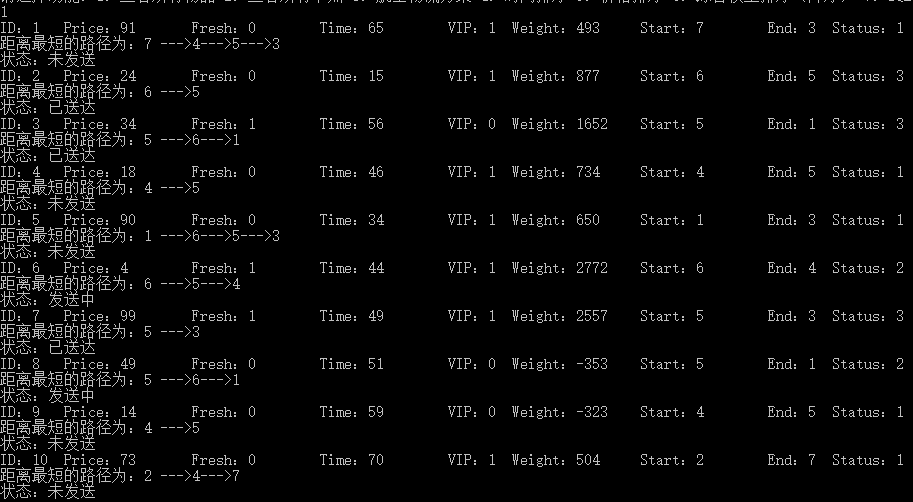
主界面：



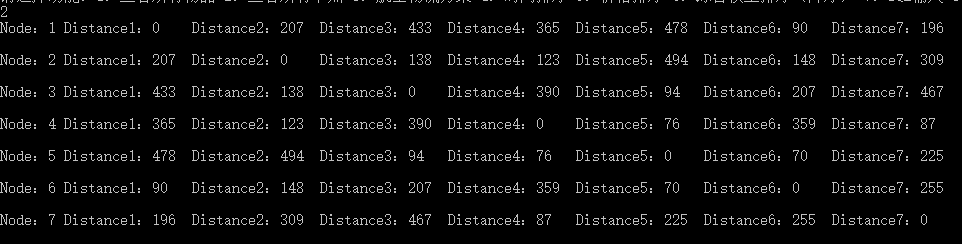
生成的文件如图：

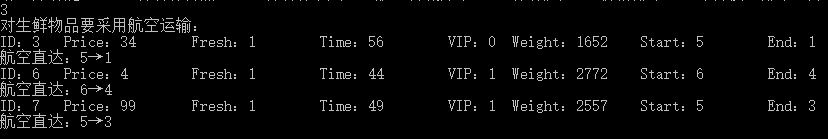
查看所有物品：



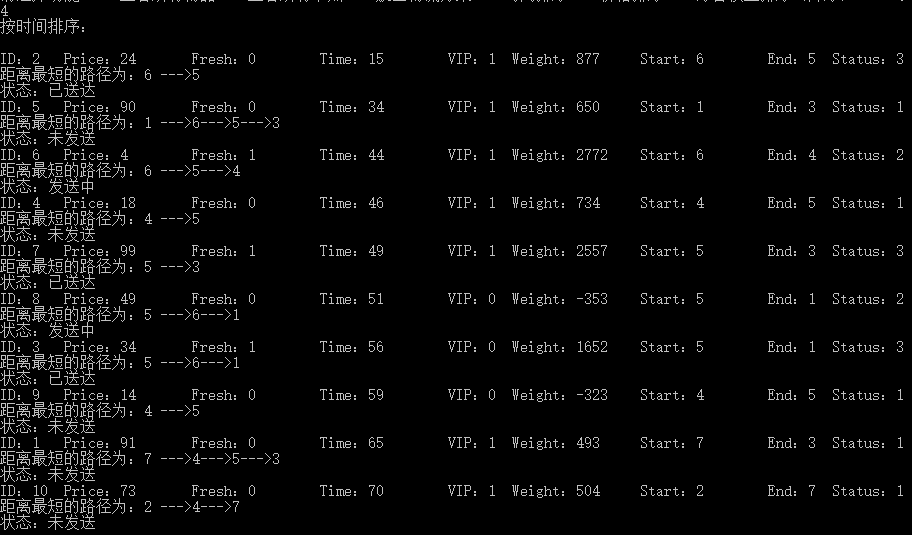
查看所有节点：



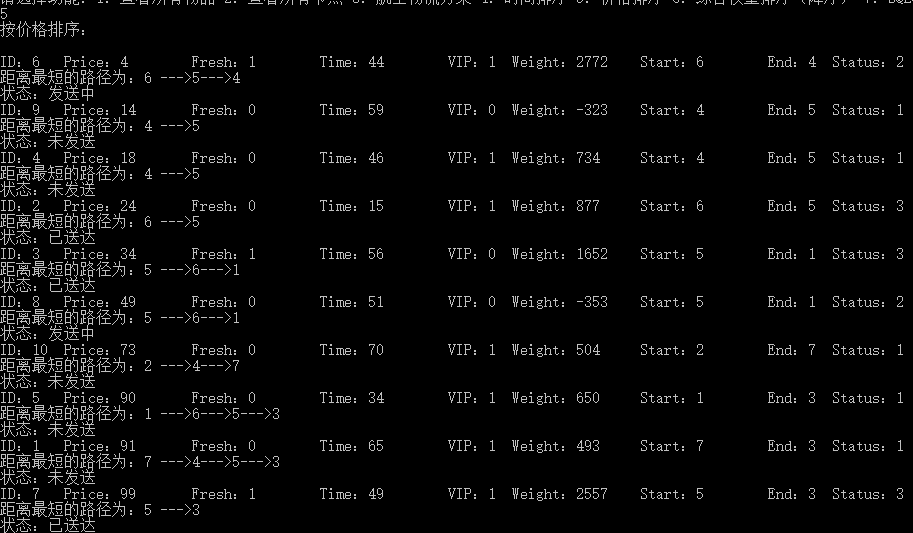
航空物流方案：



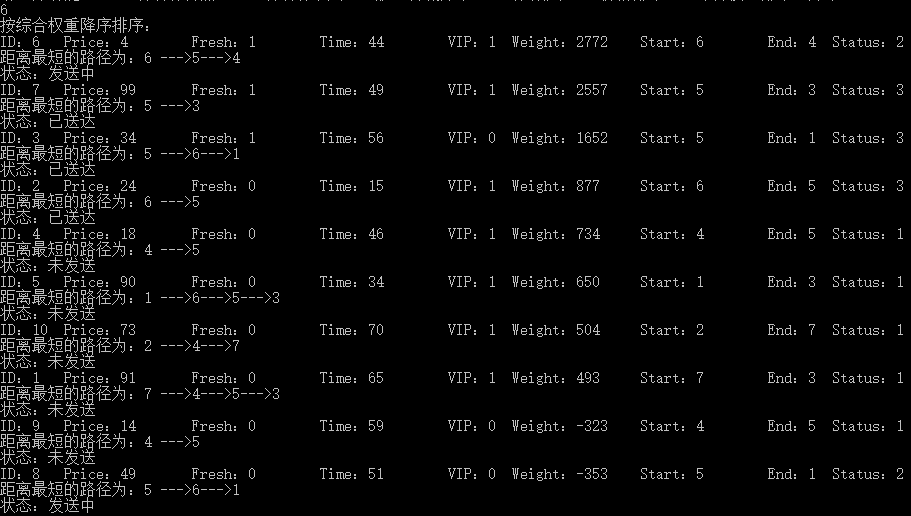
时间排序：



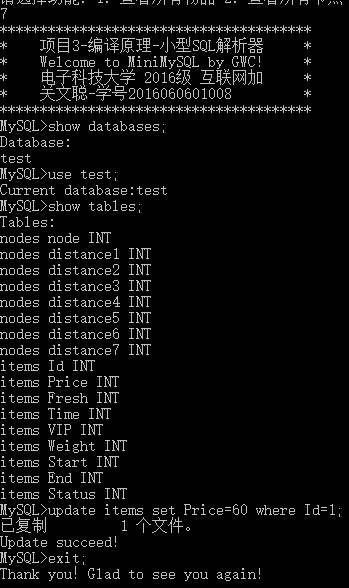
价格排序：



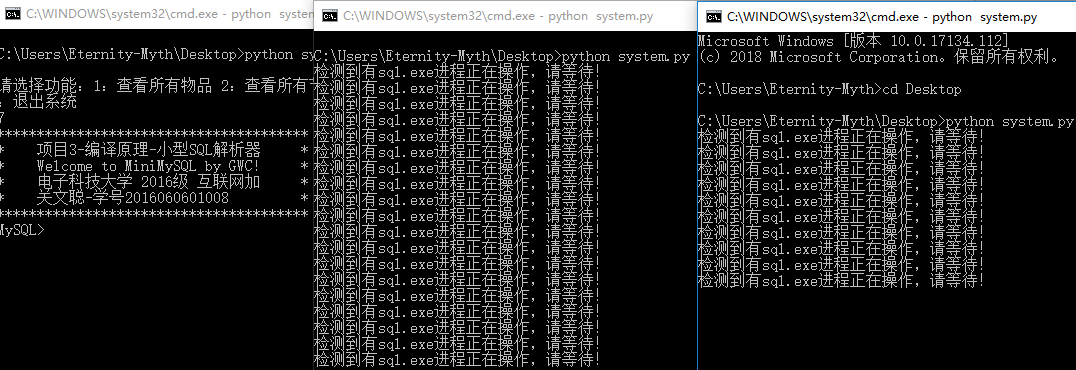
综合权重排序：



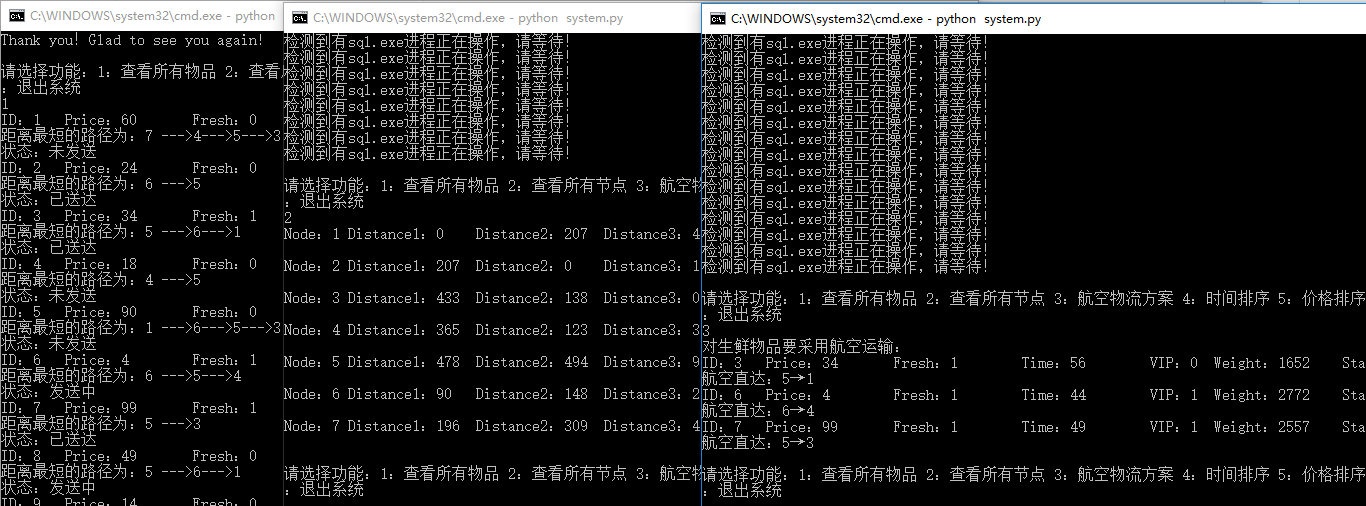
SQL解释器：



多进程并发访问读写控制（有sql.exe进程时，阻塞其它进程，其它操作需等待）：



多进程并行查询数据：



总结：

该系统较为完整地实现了实验1-4的要求和功能，完成了包括物品信息、节点信息的生成以及相应的增删改查操作，完成了简单的排序与输出操作。设计并实现了简单的SQL语句解析器，并能够对接文件操作数据。利用多进程并发访问的互斥与同步控制，保证了数据的一致性。