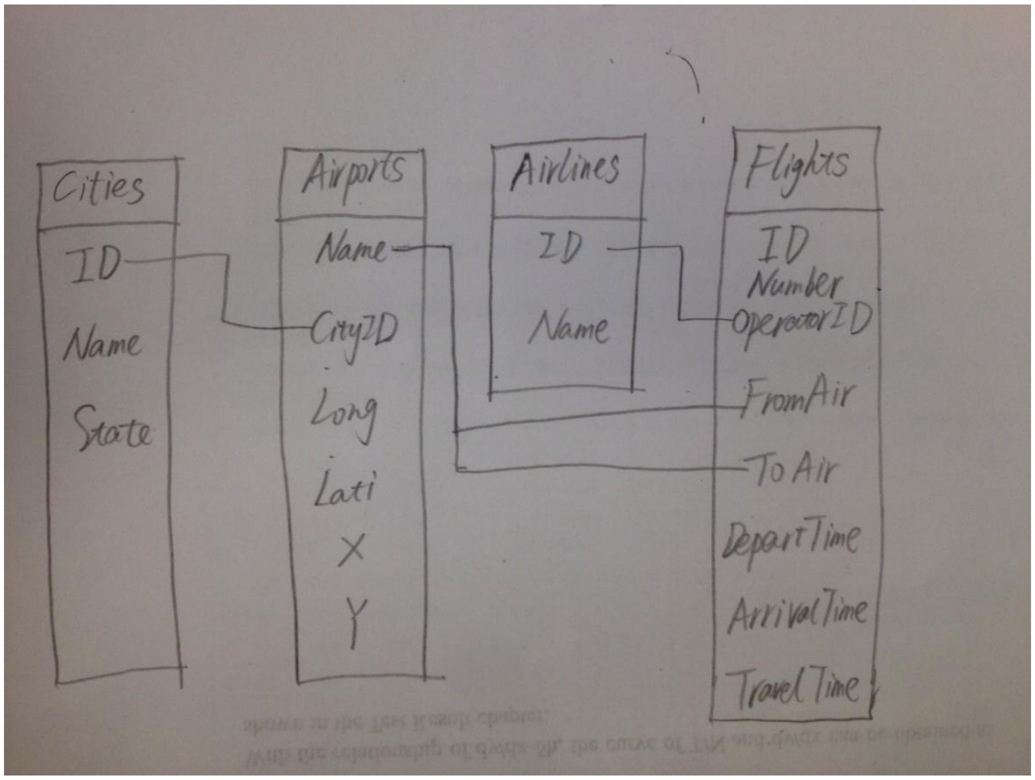
Contents

1. Part 1 Database Representation Design

1.1 Tables structure in database



1.2 Create database commands from the ‘createGraph.sql’ file

```
.open Graph.db
drop table if exists Cities;
drop table if exists Airports;
drop table if exists Airlines;
drop table if exists Flights;


create table Cities(
        ID              int not null primary key,
        name            char(20),
        state           char(5)
);

create table Airports(
        name            char(3) not null primary key,
        cityID   int,
        Long            double,
        Lati            double,
        X               double,
        Y               double
);

create table Airlines(
        ID              int not null primary key,
        name            char(20)
);

create table Flights(
        ID              int not null primary key,
        Number          char(10),
        OperatorID      int,
        FromAir         char(3),
        ToAir   char(3),
        DepartTime      int,
        ArrivalTime     int,
        TravelTime      int
);
```

1.3 Program that generates the database in the ‘CreateGraph.py’ file

```
import sqlite3 as dbi
import sys

#global counters for unique ID for cityID, AirlineID, FlightID
CTID=0
ALID=0
FLID=0

def getCityID(db, name, State):
   cu=db.cursor()
   cmd="""SELECT ID FROM Cities
         WHERE Name='{}' AND State='{}'"""
   cu.execute(cmd.format(name, State))
   cityID=cu.fetchone()
   return cityID

def createCityEntry(db,ID,name,State):
   c=getCityID(db, name, State)
```

```
    cu = db.cursor()
    sql_command = """INSERT INTO Cities (ID, Name, State)
                  VALUES ({}, '{}', '{}')"""
    if c==None :
        cu.execute(sql_command.format(ID,name,State))

def createAirportEntry(db,Name, CityID, lon, lati, x, y):
    cu = db.cursor()
    sql_command = """INSERT INTO Airports (Name, CityID, Long, Lati, X, Y)
                  VALUES ('{}', {}, {}, {}, {}, {})"""
    #print sql_command.format(ID,name,dob)
    cu.execute(sql_command.format(Name, CityID, lon, lati, x, y))

def getAirlineID(db, name):
    cu=db.cursor()
    cmd="""SELECT ID FROM Airlines
        WHERE Name='{}'"""
    cu.execute(cmd.format(name))
    AirID=cu.fetchone()
    return AirID

def createAirlineEntry(db,ID,name):
    a=getAirlineID(db, name)
    sql_command = """INSERT INTO Airlines (ID, Name)
                  VALUES ({}, '{}')"""
    cu = db.cursor()
    if a == None:
        cu.execute(sql_command.format(ID,name))

def createFlightEntry(db, ID, Number, OperatorID, FromAir, ToAir, DepartTime, ArrivalTime,
TravelTime):
    cu=db.cursor()
    sql_command="""INSERT INTO Flights (ID, Number, OperatorID, FromAir, ToAir,
DepartTime, ArrivalTime, TravelTime)
                  VALUES ({}, '{}', {}, '{}', '{}', {}, {}, {})"""
    cu.execute(sql_command.format(ID, Number, OperatorID, FromAir, ToAir, DepartTime,
ArrivalTime, TravelTime))

filename="Airport Data.txt"
#connect to the database
```

```
try:
    db = dbi.connect('Graph.db')
    print 'success'
except:
    print 'failed'
    sys.exit()
# open the file
try:
    f = open(filename,'r')
    print 'success'
except IOError:
    print 'failed when open Airport Data'
    sys.exit()


c=db.cursor()
head=f.readline().split('\t')

for line in f:
    line=line.split('\t')
    for i in range(3,len(line)):
        line[i] = float(line[i])
    CTID+=1
    createCityEntry(db, CTID, line[1], line[2])
    #find the cityID and store in cID
    sql_cmd="""SELECT ID From Cities
             WHERE Name= '{}' and State = '{}'"""
    c.execute(sql_cmd.format(line[1], line[2]))
    cID=c.fetchone()[0]
    createAirportEntry(db, line[0], cID, line[3], line[4], line[5], line[6])
f.close()

filename1="Flight Data.txt"
# open the file
try:
    g = open(filename1,'r')
    print 'success'
except IOError:
    print 'failed when open Flight Data'
    sys.exit()
head=g.readline().split('\t')
#print head



for line in g:
    line=line.split('\t')
    FLID+=1
    ALID+=1
    createAirlineEntry(db, ALID, line[1])
    #find the airlineID and store in oID
    sql_cmd="""SELECT ID From Airlines
             WHERE Name= '{}'"""
    c.execute(sql_cmd.format(line[1]))
    oID=c.fetchone()[0]
    #get the hour and minutes of departTime and arrivalTime and calculate the TravelTime
    deTime=line[4].split(':')
    arTime=line[5].split(':')
    deTime=int(deTime[0])*60+int(deTime[1])
    arTime=int(arTime[0])*60+int(arTime[1])
    TrvlTime=arTime-deTime
    createFlightEntry(db, FLID, line[0], oID, line[2], line[3], deTime, arTime,TrvlTime)

db.commit()
db.close()
```

2. Part 2 Path search algorithm adaption

2.1 Graph.py

```
import sqlite3 as dbi
import sys
from copy import deepcopy

class Graph (object):
    def __init__(self, foldername):
        try:
            self.db = dbi.connect(foldername)
            self.cu=self.db.cursor()
            print 'success'
        except:
            print 'failed'
            sys.exit()


    #return a list of flights.ID going out from the cityName
    def getAttachedLines(self, aptName):
        cmd="""SELECT ID FROM Flights
              WHERE FromAir='{}'"""
        self.cu.execute(cmd.format(aptName))
        l=[]
        for li in self.cu.fetchall():
            l.append(li[0])
        return l
    #return the cities.Name reached by the flightsID
    def getAttachedNodes(self, fltID):
        cmd="""SELECT ToAir FROM Flights
              WHERE ID='{}' """
        self.cu.execute(cmd.format(fltID))
        return self.cu.fetchone()
    #helper function getting city and state of airport from airportName
    def getCityofAirport(self,AptName):
        cmdgetcity="""SELECT c.Name, c.State FROM Cities AS c, Airports AS a
                  WHERE a.cityID=c.ID AND A.Name='{}'"""
        self.cu.execute(cmdgetcity.format(AptName))
        return self.cu.fetchone()
    #helper function getting necessary information of flight for given flight ID and return a string in a
format of print
    def printFlightInfo(self, fltID):

        cmdgetflightinfo="""SELECT f.Number, fc.Name, fc.State, fa.Name, f.DepartTime, tc.Name,
tc.State, ta.Name, f.ArrivalTime
                        FROM Flights AS f, Airlines AS al, Airports AS fa, Airports AS ta, Cities AS fc, Cities
AS tc
                        WHERE al.ID=f.OperatorID AND f.FromAir=fa.Name AND fa.cityID=fc.ID AND
f.ToAir=ta.Name AND ta.CityID=tc.ID
                            AND f.ID={}"""
        self.cu.execute(cmdgetflightinfo.format(fltID))
        fl = self.cu.fetchone()
        s="""on Flight Number {} from {}, {}, ({}) at {}:{:02d}, arrive at {}, {} ({}) at {}:{:02d} \n"""\
             .format(fl[0], fl[1],fl[2],fl[3],fl[4]/60,fl[4]%60,fl[5], fl[6], fl[7], fl[8]/60,fl[8]%60)
        return s
    #helper function getting necessary information of flight for given flight ID and return a list of the infos
rather than string
    def FlightInfo(self, fltID):
        cmdgetflightinfo="""SELECT f.Number, fc.Name, fc.State, fa.Name, f.DepartTime, tc.Name,
tc.State, ta.Name, f.ArrivalTime
                        FROM Flights AS f, Airlines AS al, Airports AS fa, Airports AS ta, Cities AS fc, Cities
AS tc
                        WHERE al.ID=f.OperatorID AND f.FromAir=fa.Name AND fa.cityID=fc.ID AND
f.ToAir=ta.Name AND ta.CityID=tc.ID
                            AND f.ID={}"""
        self.cu.execute(cmdgetflightinfo.format(fltID))
        return self.cu.fetchone()

    def __str__(self):
        cmd="""SELECT c.Name, a.Name, a.X, a.Y
            FROM Airports AS a, Cities AS c
            WHERE a.cityID=c.ID"""
        self.cu.execute(cmd)
        allApts=self.cu.fetchall()
        s='*Airports: \n'
        for apt in allApts:
            s+="""{} : {} at ({} miles, {} miles) \n""".format(apt[0], apt[1], apt[2], apt[3])

        s+='* Flights:\n'

        cmd="""SELECT f.Number, al.Name, fc.Name, fc.State, fa.Name, f.DepartTime, tc.Name, tc.State,
ta.Name, f.ArrivalTime
            FROM Flights AS f, Airlines AS al, Airports AS fa, Airports AS ta, Cities AS fc, Cities AS tc
            WHERE al.ID=f.OperatorID AND f.FromAir=fa.Name AND fa.cityID=fc.ID AND
f.ToAir=ta.Name AND ta.CityID=tc.ID"""
        self.cu.execute(cmd)
        allFlights=self.cu.fetchall()

        for fl in allFlights:
            s+="""Flight Number {} operated by {}: leaving {}, {}, ({}) at {}:{:02d} to {}, {} ({}) arriving at
{}:{:02d} \n"""\
                .format(fl[0], fl[1],fl[2],fl[3],fl[4],fl[5]/60,fl[5]%60,fl[6], fl[7], fl[8], fl[9]/60,fl[9]%60)
        return s
```

```
def findPath(self, startName, endName):
    cmdtogetapt = """SELECT Name FROM Airports"""
    self.cu.execute(cmdtogetapt)
    aptList = self.cu.fetchall()
    apts=[]
    for l in aptList:
        apts.append(l[0])

    if startName not in apts :
        print "unknown AirportName=%s" % startName
    if endName not in apts :
        print "unknown AirportName=%s" % endName
    global pathlist
    (traveledLines, traveledNodes)=([],[])
    path= dict (nodepath=[], linepath=[], length= 0.0)
    pathlist=[]
    startName=str(startName)
    endName=str(endName)
    self.findPathInside(startName,endName, path,traveledNodes[:], traveledLines[:])

    return pathlist

def findPathInside(self, startName, endName, path, GtraveledNodes, GtraveledLines):

    if startName not in GtraveledNodes:
        #traveledNodes=GtraveledNodes[:]
        GtraveledNodes.append(startName)



        cmdtogettrvltime = """SELECT DepartTime, ArrivalTime  FROM Flights
                              WHERE ID='{}'"""
        #get all the path has went through
        lastpath=path['linepath']
        if len(lastpath) >0:
            #get the time information for the previous path has travelled
            self.cu.execute(cmdtogettrvltime.format(lastpath[len(lastpath)-1]))
            lastDETime = self.cu.fetchone()[0]
            self.cu.execute(cmdtogettrvltime.format(lastpath[len(lastpath)-1]))
            lastARTime = self.cu.fetchone()[1]
        else:
            lastARTime=-1

        for l in self.getAttachedLines(startName):
            self.cu.execute(cmdtogettrvltime.format(l))
            newDETime= self.cu.fetchone()[0]
            self.cu.execute(cmdtogettrvltime.format(l))
            newARTime = self.cu.fetchone()[1]

            traveledNodes = GtraveledNodes[:]
            traveledLines = GtraveledLines[:]

            if (l not in traveledLines):
                traveledLines.append(l)
                #print type(l)
                #print type(self.getAttachedNodes(l))
                for n in  self.getAttachedNodes(l):
                    if n not in traveledNodes:
                        pathInside=deepcopy(path)
                        pathInside['nodepath'].append(startName)
                        pathInside['linepath'].append(l)
                        if lastARTime==-1:
                            T1=0
                        else:
                            T1=newDETime-lastARTime
                            if T1 <=0 :
                                T1=T1+24*60
                        pathInside['length']+=T1+newARTime-newDETime
                        if n==endName:
                            pathInside['nodepath'].append(endName)
                            pathlist.append(pathInside)
                        else:
                            self.findPathInside(n, endName, pathInside,traveledNodes, traveledLines)


    def findShortestPath(self, startName, endName):
        allPath=self.findPath(startName, endName)
        #print allPath
        shortestLen=allPath[0]['length']
        j=0
        for i in range(1,len(allPath)):
            if allPath[i]['length']<shortestLen:
                j=i
                shortestLen=allPath[i]['length']
        #do all the strings for the print
        #get the start, end airport information and the first and last flight in the path for the first print line
        spath=allPath[j]
        startInfo=self.getCityofAirport(startName)
        endInfo=self.getCityofAirport(endName)
        firstline=self.FlightInfo(spath['linepath'][0])
        lastline=self.FlightInfo(spath['linepath'][len(spath['linepath'])-1])
        s='Trip: {}: {}, {} to {}: {}, {}\n departs at {}:{:02d}, arrives at {}:{:02d} after travelling for
{}:{:02d} hours\n'\
           .format(startName, startInfo[0], startInfo[1], endName, endInfo[0], endInfo[1], firstline[4]/60,
firstline[4]%60,\
                   lastline[8]/60, lastline[8]%60, int(spath['length'])/60, int(spath['length'])%60)
        #add each flight information to the printout string
        for f in spath['linepath']:
            s+=self.printFlightInfo(f)

        return s



    def numNodes(self):
        cmd="""SELECT count(ID) FROM Flights"""
        self.cu.execute(cmd)
        return self.cu.fetchone()[0]

    def numLines(self):
        cmd="""SELECT count(ID) FROM Flights"""
        self.cu.execute(cmd)
        return self.cu.fetchone()[0]
```

How to use, test code

3. Uses, tests, outcomes and answers

3.1 How to use the code

3.1.1 To create the database

First, read the ‘createGraph.sql’ file in Sqlite3, which creates a new graph with the designed tables and structure.

Then, run the 'CreateGraph.py' file in python, which connects to the graph database and stores the data given in the excel file in the database.

Through the main function which creates a Graph class, one can get data from the database and find shortest path from one airport to another one. The result of the shortest path is printed from the string returned.

I tested my code by the ways above and the procedures and results are shown below.

3.2 Test 1

I tried to find the shortest path from 'SFO' to 'LGA', and the output is same as professor posted online.

3.1.1 Test method in 'main.py'

```
from Graph import *
g = Graph('Graph.db')
nodeIDi='SFO'
nodeIDj='LGA'
shortpath=g.findShortestPath( nodeIDj, nodeIDi)
print shortpath
```

3.1.2 Output:

```
success
Trip: LGA: New York, NY to SFO: San Francisco, CA
 departs at 11:30, arrives at 18:15 after travelling for 6:45 hours
on Flight Number UA345 from New York, NY, (LGA) at 11:30, arrive at Chicago, IL (ORD) at 14:05
on Flight Number UA49 from Chicago, IL, (ORD) at 14:50, arrive at San Francisco, CA (SFO) at 18:15
```

3.3 Test 2 Answer for Seattle to Miami

3.2.1Method in 'main.py'

```
from Graph import *

g = Graph('Graph.db')
#print g

nodeIDi='SEA'
nodeIDj='MIA'
shortpath=g.findShortestPath( nodeIDi, nodeIDj)
print shortpath
```

3.2.2 Answer:

```
success
Trip: SEA: Seattle, WA to MIA: Miami, FL
 departs at 9:45, arrives at 17:05 after travelling for 7:20 hours
```

```
on Flight Number SWA123 from Seattle, WA, (SEA) at 9:45, arrive at Denver, CO (DIA) at 11:55
on Flight Number SWA125 from Denver, CO, (DIA) at 13:25, arrive at Miami, FL (MIA) at 17:05
```

3.4 Test 3 Answer for Miami to Seattle

3.3.1Method in ‘main.py’

```
from Graph import *

g = Graph('Graph.db')
#print g

nodeIDi='SEA'
nodeIDj='MIA'
shortpath=g.findShortestPath( nodeIDj, nodeIDi)
print shortpath
```

3.3.2 Answer:

```
 success
Trip: MIA: Miami, FL to SEA: Seattle, WA
 departs at 8:10, arrives at 18:05 after travelling for 9:55 hours
on Flight Number AA1302 from Miami, FL, (MIA) at 8:10, arrive at Dallas, TX (DFW) at 10:40
on Flight Number DL1214 from Dallas, TX, (DFW) at 12:40, arrive at Salt Lake City, UT (SLC) at 14:05
on Flight Number DL2222 from Salt Lake City, UT, (SLC) at 15:10, arrive at Seattle, WA (SEA) at
18:05
```

3.5 Test 4 Print out Test

3.4.1Method in ‘main.py’

```
from Graph import *
g = Graph('Graph.db')
print g
```

3.4.2 Output

```
success
*Airports:
Seattle : SEA at (1652.969 miles, 3289.715 miles)
San Francisco : SFO at (1933.199 miles, 2610.735 miles)
Denver : DIA at (2806.848 miles, 2745.883 miles)
Oakland : OAK at (1940.612 miles, 2612.193 miles)
New York : JFK at (4406.324 miles, 2808.105 miles)
Miami : MIA at (4828.976 miles, 1781.011 miles)
Dallas : DFW at (3545.021 miles, 2264.791 miles)
New York : LGA at (4392.332 miles, 2817.606 miles)
```

Salt Lake City : SLC at (2404.426 miles, 2815.727 miles)
Chicago : ORD at (3591.154 miles, 2900.62 miles)
* Flights:
Flight Number CB1 operated by City Bus: leaving New York, NY, (LGA) at 6:00 to New York, NY (JFK) arriving at 7:00
Flight Number AK2155 operated by Alaska: leaving Seattle, WA, (SEA) at 6:45 to Denver, CO (DIA) arriving at 8:55
Flight Number CBA operated by City Bus: leaving New York, NY, (JFK) at 7:00 to New York, NY (LGA) arriving at 8:00
Flight Number AK1256 operated by Alaska: leaving Seattle, WA, (SEA) at 7:15 to Chicago, IL (ORD) arriving at 10:55
Flight Number CB2 operated by City Bus: leaving New York, NY, (LGA) at 8:00 to New York, NY (JFK) arriving at 9:00
Flight Number UA768 operated by Unites Airlines: leaving Miami, FL, (MIA) at 8:05 to Dallas, TX (DFW) arriving at 10:49
Flight Number AA1302 operated by American Airlines: leaving Miami, FL, (MIA) at 8:10 to Dallas, TX (DFW) arriving at 10:40
Flight Number DL34 operated by Delta: leaving Seattle, WA, (SEA) at 8:30 to Salt Lake City, UT (SLC) arriving at 11:45
Flight Number SWA10 operated by Southwest: leaving Seattle, WA, (SEA) at 8:45 to Oakland, CA (OAK) arriving at 11:15
Flight Number UA184 operated by Unites Airlines: leaving Miami, FL, (MIA) at 8:50 to Chicago, IL (ORD) arriving at 12:30
Flight Number CBB operated by City Bus: leaving New York, NY, (JFK) at 9:00 to New York, NY (LGA) arriving at 10:00
Flight Number AK3392 operated by Alaska: leaving Seattle, WA, (SEA) at 9:10 to San Francisco, CA (SFO) arriving at 11:25
Flight Number AK1234 operated by Alaska: leaving Denver, CO, (DIA) at 9:40 to Seattle, WA (SEA) arriving at 12:05
Flight Number SWA123 operated by Southwest: leaving Seattle, WA, (SEA) at 9:45 to Denver, CO (DIA) arriving at 11:55
Flight Number AA482 operated by American Airlines: leaving Miami, FL, (MIA) at 9:45 to New York, NY (LGA) arriving at 11:55
Flight Number CB3 operated by City Bus: leaving New York, NY, (LGA) at 10:00 to New York, NY (JFK) arriving at 11:00
Flight Number CBC operated by City Bus: leaving New York, NY, (JFK) at 11:00 to New York, NY (LGA) arriving at 12:00
Flight Number UA123 operated by Unites Airlines: leaving Seattle, WA, (SEA) at 11:00 to New York, NY (JFK) arriving at 17:20

Flight Number AA345 operated by American Airlines: leaving Chicago, IL, (ORD) at 11:25 to New York, NY (JFK) arriving at 13:55
Flight Number UA345 operated by Unites Airlines: leaving New York, NY, (LGA) at 11:30 to Chicago, IL (ORD) arriving at 14:05
Flight Number SWA11 operated by Southwest: leaving Oakland, CA, (OAK) at 11:45 to Seattle, WA (SEA) arriving at 14:15
Flight Number AA92 operated by American Airlines: leaving Dallas, TX, (DFW) at 11:55 to New York, NY (LGA) arriving at 14:20
Flight Number DL882 operated by Delta: leaving Oakland, CA, (OAK) at 11:55 to Dallas, TX (DFW) arriving at 15:10
Flight Number CB4 operated by City Bus: leaving New York, NY, (LGA) at 12:00 to New York, NY (JFK) arriving at 13:00
Flight Number AK3245 operated by Alaska: leaving San Francisco, CA, (SFO) at 12:15 to Seattle, WA (SEA) arriving at 14:50
Flight Number DL382 operated by Delta: leaving Salt Lake City, UT, (SLC) at 12:35 to Chicago, IL (ORD) arriving at 14:55
Flight Number DL1214 operated by Delta: leaving Dallas, TX, (DFW) at 12:40 to Salt Lake City, UT (SLC) arriving at 14:05
Flight Number CBD operated by City Bus: leaving New York, NY, (JFK) at 13:00 to New York, NY (LGA) arriving at 14:00
Flight Number AK2241 operated by Alaska: leaving Dallas, TX, (DFW) at 13:05 to Denver, CO (DIA) arriving at 14:50
Flight Number SWA125 operated by Southwest: leaving Denver, CO, (DIA) at 13:25 to Miami, FL (MIA) arriving at 17:05
Flight Number DL1212 operated by Delta: leaving Salt Lake City, UT, (SLC) at 13:40 to Dallas, TX (DFW) arriving at 15:10
Flight Number CB5 operated by City Bus: leaving New York, NY, (LGA) at 14:00 to New York, NY (JFK) arriving at 15:00
Flight Number SWA76 operated by Southwest: leaving Oakland, CA, (OAK) at 14:00 to Dallas, TX (DFW) arriving at 16:38
Flight Number UA76 operated by Unites Airlines: leaving New York, NY, (JFK) at 14:10 to Miami, FL (MIA) arriving at 16:05
Flight Number UA49 operated by Unites Airlines: leaving Chicago, IL, (ORD) at 14:50 to San Francisco, CA (SFO) arriving at 18:15
Flight Number AA562 operated by American Airlines: leaving Chicago, IL, (ORD) at 14:50 to Dallas, TX (DFW) arriving at 17:40
Flight Number AA734 operated by American Airlines: leaving New York, NY, (LGA) at 14:55 to Denver, CO (DIA) arriving at 17:45
Flight Number DL885 operated by Delta: leaving Chicago, IL, (ORD) at 14:55 to Salt Lake City, UT (SLC) arriving at 17:05

```
Flight Number CBE operated by City Bus: leaving New York, NY, (JFK) at 15:00 to New York, NY
(LGA) arriving at 16:00
Flight Number DL2222 operated by Delta: leaving Salt Lake City, UT, (SLC) at 15:10 to Seattle, WA
(SEA) arriving at 18:05
Flight Number CB6 operated by City Bus: leaving New York, NY, (LGA) at 16:00 to New York, NY
(JFK) arriving at 17:00
Flight Number AK246 operated by Alaska: leaving Denver, CO, (DIA) at 16:10 to Seattle, WA (SEA)
arriving at 18:40
Flight Number AA281 operated by American Airlines: leaving New York, NY, (LGA) at 16:15 to Miami,
FL (MIA) arriving at 18:35
Flight Number AK3398 operated by Alaska: leaving Seattle, WA, (SEA) at 16:40 to San Francisco, CA
(SFO) arriving at 19:00
Flight Number CBF operated by City Bus: leaving New York, NY, (JFK) at 17:00 to New York, NY
(LGA) arriving at 18:00
Flight Number UA234 operated by Unites Airlines: leaving Salt Lake City, UT, (SLC) at 17:25 to San
Francisco, CA (SFO) arriving at 19:50
Flight Number DL421 operated by Delta: leaving Miami, FL, (MIA) at 17:40 to Denver, CO (DIA)
arriving at 21:20
Flight Number CB7 operated by City Bus: leaving New York, NY, (LGA) at 18:00 to New York, NY
(JFK) arriving at 19:00
Flight Number AA0123 operated by American Airlines: leaving Dallas, TX, (DFW) at 18:35 to Miami,
FL (MIA) arriving at 20:50
Flight Number CBG operated by City Bus: leaving New York, NY, (JFK) at 19:00 to New York, NY
(LGA) arriving at 20:00
Flight Number AK3248 operated by Alaska: leaving San Francisco, CA, (SFO) at 19:45 to Seattle, WA
(SEA) arriving at 21:55
```

4. Select statement and output

SELECT Statements used in the code are listed. Some statements which need parameters passed in from the program cannot be tested separately and thus not listed here, but the functionality of those statements are verified in the program outputs.

4.1 SELECT Statement 1

Statement:

```
SELECT c.Name, a.Name, a.X, a.Y
        FROM Airports AS a, Cities AS c
        WHERE a.cityID=c.ID
```

Output:

```
name|name|X|Y
Seattle|SEA|1652.969|3289.715
San Francisco|SFO|1933.199|2610.735
```

```
Denver|DIA|2806.848|2745.883
Oakland|OAK|1940.612|2612.193
New York|JFK|4406.324|2808.105
Miami|MIA|4828.976|1781.011
Dallas|DFW|3545.021|2264.791
New York|LGA|4392.332|2817.606
Salt Lake City|SLC|2404.426|2815.727
Chicago|ORD|3591.154|2900.62
```

4.2 SELECT Statement 2

Statement:

```
SELECT f.Number, al.Name, fc.Name, fc.State, fa.Name, f.DepartTime, tc.Name, tc.State, ta.Name,
f.ArrivalTime
        FROM Flights AS f, Airlines AS al, Airports AS fa, Airports AS ta, Cities AS fc, Cities AS tc
        WHERE al.ID=f.OperatorID AND f.FromAir=fa.Name AND fa.cityID=fc.ID AND
f.ToAir=ta.Name AND ta.CityID=tc.ID
```

Output:

```
Number|name|name|state|name|DepartTime|name|state|name|ArrivalTime
CB1|City Bus|New York|NY|LGA|360|New York|NY|JFK|420
AK2155|Alaska|Seattle|WA|SEA|405|Denver|CO|DIA|535
CBA|City Bus|New York|NY|JFK|420|New York|NY|LGA|480
AK1256|Alaska|Seattle|WA|SEA|435|Chicago|IL|ORD|655
CB2|City Bus|New York|NY|LGA|480|New York|NY|JFK|540
UA768|Unites Airlines|Miami|FL|MIA|485|Dallas|TX|DFW|649
AA1302|American Airlines|Miami|FL|MIA|490|Dallas|TX|DFW|640
DL34|Delta|Seattle|WA|SEA|510|Salt Lake City|UT|SLC|705
SWA10|Southwest|Seattle|WA|SEA|525|Oakland|CA|OAK|675
UA184|Unites Airlines|Miami|FL|MIA|530|Chicago|IL|ORD|750
CBB|City Bus|New York|NY|JFK|540|New York|NY|LGA|600
AK3392|Alaska|Seattle|WA|SEA|550|San Francisco|CA|SFO|685
AK1234|Alaska|Denver|CO|DIA|580|Seattle|WA|SEA|725
SWA123|Southwest|Seattle|WA|SEA|585|Denver|CO|DIA|715
AA482|American Airlines|Miami|FL|MIA|585|New York|NY|LGA|715
CB3|City Bus|New York|NY|LGA|600|New York|NY|JFK|660
CBC|City Bus|New York|NY|JFK|660|New York|NY|LGA|720
UA123|Unites Airlines|Seattle|WA|SEA|660|New York|NY|JFK|1040
AA345|American Airlines|Chicago|IL|ORD|685|New York|NY|JFK|835
UA345|Unites Airlines|New York|NY|LGA|690|Chicago|IL|ORD|845
```

```
SWA11|Southwest|Oakland|CA|OAK|705|Seattle|WA|SEA|855
AA92|American Airlines|Dallas|TX|DFW|715|New York|NY|LGA|860
DL882|Delta|Oakland|CA|OAK|715|Dallas|TX|DFW|910
CB4|City Bus|New York|NY|LGA|720|New York|NY|JFK|780
AK3245|Alaska|San Francisco|CA|SFO|735|Seattle|WA|SEA|890
DL382|Delta|Salt Lake City|UT|SLC|755|Chicago|IL|ORD|895
DL1214|Delta|Dallas|TX|DFW|760|Salt Lake City|UT|SLC|845
CBD|City Bus|New York|NY|JFK|780|New York|NY|LGA|840
AK2241|Alaska|Dallas|TX|DFW|785|Denver|CO|DIA|890
SWA125|Southwest|Denver|CO|DIA|805|Miami|FL|MIA|1025
DL1212|Delta|Salt Lake City|UT|SLC|820|Dallas|TX|DFW|910
CB5|City Bus|New York|NY|LGA|840|New York|NY|JFK|900
SWA76|Southwest|Oakland|CA|OAK|840|Dallas|TX|DFW|998
UA76|Unites Airlines|New York|NY|JFK|850|Miami|FL|MIA|965
UA49|Unites Airlines|Chicago|IL|ORD|890|San Francisco|CA|SFO|1095
AA562|American Airlines|Chicago|IL|ORD|890|Dallas|TX|DFW|1060
AA734|American Airlines|New York|NY|LGA|895|Denver|CO|DIA|1065
DL885|Delta|Chicago|IL|ORD|895|Salt Lake City|UT|SLC|1025
CBE|City Bus|New York|NY|JFK|900|New York|NY|LGA|960
DL2222|Delta|Salt Lake City|UT|SLC|910|Seattle|WA|SEA|1085
CB6|City Bus|New York|NY|LGA|960|New York|NY|JFK|1020
AK246|Alaska|Denver|CO|DIA|970|Seattle|WA|SEA|1120
AA281|American Airlines|New York|NY|LGA|975|Miami|FL|MIA|1115
AK3398|Alaska|Seattle|WA|SEA|1000|San Francisco|CA|SFO|1140
CBF|City Bus|New York|NY|JFK|1020|New York|NY|LGA|1080
UA234|Unites Airlines|Salt Lake City|UT|SLC|1045|San Francisco|CA|SFO|1190
DL421|Delta|Miami|FL|MIA|1060|Denver|CO|DIA|1280
CB7|City Bus|New York|NY|LGA|1080|New York|NY|JFK|1140
AA0123|American Airlines|Dallas|TX|DFW|1115|Miami|FL|MIA|1250
CBG|City Bus|New York|NY|JFK|1140|New York|NY|LGA|1200
AK3248|Alaska|San Francisco|CA|SFO|1185|Seattle|WA|SEA|1315
```

4.3 SELECT Statement 3

Statement:

```
SELECT Name FROM Airports
```

Output:

```
name
DFW
```

```
DIA
JFK
LGA
MIA
OAK
ORD
SEA
SFO
SLC
```

4.4 SELCT Statement 5

Statement:

```
SELECT count(Name) FROM Airports
```

Output:

```
count(Name)
10
```

4.5 SELECT Statement 4

Statement:

```
SELECT count(ID) FROM Flights
```

Output:

```
count(ID)
51
```

5. Key features

My implementation first creates the database through sqlite by reading a file which has the commands to create the database. Then read the input file by a separate python program, which connects the existed database and write the data to the database. After that, the python program ‘Graph.py’ creates a graph class which interacts with the database. Thus through create Graph class instance, the client program ‘Main’ could read data in the database, and find shorted paths. The key features is to connect the database with a python class, and use the previous algorithm to find shortest path merely by changing some methods to get data from database rather than storing them in python variabls.