使用uiautomation和路路通PC端程序生成时刻表

目前路路通PC段和移动端当前数据库均不能提供时刻表中的检票口信息，无法得知具体股道故使用映射

当前状态

- 在PC端使用本地数据库其效率高于爬虫但是过分依赖离线数据库自身
  - 处理效率较高，一分钟可生成50条时刻表数据
    - 这里使用生产者-消费者模型，生产者获取离线数据库数据，消费者生成时刻表
    - 使用队列来传递数据，消费者速度远快于生产者，考虑多开生产者线程
  - 对于发布的离线数据库依赖较大
    - 信息准确性高，易于生成径路等数据
    - 目前版本数据库没有检票口数据，导致只能使用映射
    - 在早先版本存在检票口数据时获取的数据质量不佳处理难度相对较大(存在数据黏连)
- 目前爬取效果,处理等和前面武汉枢纽相同
  - 基本实现自动化,需要修改的数据主要为游戏车站信息，线路信息用于对应列车前序站和后续站以生成径路以及中心车站自身基本信息
  - 最后生成时可能会由于调图等原因(当日不开行)有多个相同车次，去重即可
- 完善度表格:

组件信息	完善程度	当前状态	日后计划
车站	完善	可以按照单一车站进行查询	暂无
车次	完善	可以获得单一车次给出的大部分信息	暂无
检票口/股道	目前数据库版本无信息， 但是已有对应处理部分	暂时使用径路规则映射站台和线路所	优化逻辑
立折车辆	纯手动，没有处理	手动查找修改	使用甘特图手动查看修改
越行车辆	纯手动，完全没有处理	没有计划	可能会依照大站来获取信息

使用的库函数信息和达成的基本效果

大体分为了三个部分

- 生产者-获取整体信息
  - 从PC端程序直接获取(全选+复制)原生字符串信息并插入队列
  - 视为一个对象的行为合并为一个类
- 消费者-数据处理/数据分析
  - 从队列获取数据并整理出时刻表
  - 部分基于之前数据处理部分，经过一定的修改
  - 视为数据的流处理按照函数处理
- 可视化--开发中
  - 生成甘特图以进行早期冲突查询

第一部分，定义使用的库函数和常量信息

内容

库函数主要分为两种：获取数据--处理数据--以及多线程用的函数。
常量部分主要用于处理数据部分，映射生成符合游戏格式的时刻表，这部分常量会随着函数的完善而更为精简
基于进场线路和立场线路生成线路所和站台信息

待完善/优化的部分

1. 暂无
2. .....

In [ ]:
```
import uiautomation as auto
import subprocess
import datetime
import time
import itertools
import multiprocessing
from multiprocessing.queues import Queue
import numpy
import pandas
import random


# 速度和编组以及类型映射关系,0为普速1为动车2为高速，后续修改
species1 = {'K': ['120', 'LPPPPPPP', "0"], 'T': ['140', 'LPPPPPPP', "0"], 'Z': [
            'D': ['250', 'LLPPPPLL', "1"], 'C': ['200', 'LPPL', "1"], 'G': ['350
species1default = ['120', 'LCPPPPPPP', "0"]
# 车站-编号,掉向,用时以及运行车辆种类映射关系
# 图片左(0)右(1)侧线路key值相同则掉向,
# 国铁车辆行走左侧,2为数据为左侧股道编号

# [车站编号,车站所在侧(没用了已经),车辆进场股道,车辆离场行走股道,到达中心车站所用时间]
gameStationInfo = {'武汉站': ['a', -1, 0, 0, 0], '动车所': ['b', 0, 0, 0, 20],
                   '汉口': ['e', 0, 2, 1, 5],
                   '孝感北': ['d', 0, 1, 2, 4], '红安西': ['k', -1, 1, 2, 4],
                   '咸宁北': ['c', 0, 2, 1, 3],
                   '鄂州': ['f', -1, 2, 1, 5], '黄冈': ['g', 0, 2, 1, 5],
```

```
                    '葛店南站': ['葛店南', -1, 0, 0, 5],
                    }
gameStationDefault = ['未知', -1, 0, 0, 20]  # 找不到的默认设置

# 线路和车站关系，主要用于从车站-值获取线路-键
route1 = {'动车所': ["src", "dst"],
          '汉口': ["汉口"], '葛店南站': ["葛店南"],
          '红安西': ["红安西", "麻城北", "六安", "金寨", "合肥", "合肥南", "南京南"]
          '孝感北': ["孝感北", "信阳东", "明港东", "驻马店西", "漯河西", "许昌东", "
          '咸宁北': ["咸宁北", "赤壁北", "岳阳东", "汨罗东", "长沙南", "广州南", "广
          '鄂州': ["华容南", "鄂州", "黄石北", "大冶北", "白沙铺", "阳新", "瑞昌西",
          '黄冈': ["华容东", "黄冈西", "黄冈东", "浠水南", "蕲春西", "武穴北", "黄梅
          }
route1Default = ["src", "dst"]  # 找不到的默认设置
```

内容

暂时替代数据库原生的检票口信息部分
基于进场线路和立场线路生成线路所和站台信息
线路所为固定映射，站台为该站场随机生成，注意可达性问题

In [ ]:
```
def mapEntrRoute(src, dst, arrtime, depatime):
    # 使用同一径路的线路
    stgp1 = "fgh"  # 武九客专方向-葛店南、鄂州、黄冈
    stgp2 = "c"  # 咸宁北
    stgp3 = "dk"  # 京广高速郑州、沪蓉线合肥方向
    stgp4 = "e"  # 汉口
    arrLinePost = ""
    depaLinePost = ""
    entret = 0
    if src == "b":  # 站台目前直接随机
        entret = random.randint(1, 15)
    elif src == "e" and dst == "c":
        entret = random.randint(7, 15)
        arrLinePost = "j#1#{t}#0 ".format(t=arrtime)
    elif src == "e" and dst in stgp1:
        entret = random.randint(16, 20)
        arrLinePost = "j#2#{t}#0 ".format(t=arrtime)
        depaLinePost = "p#1#{t}#0 ".format(t=depatime)
    elif src in stgp3 and dst == "c":
        entret = random.randint(7, 15)
    elif src in stgp3 and dst in stgp1:
        entret = random.randint(7, 15)
        arrLinePost = "p#3#{t}#0 ".format(t=arrtime)

    elif src == "c" and dst == "e":
        entret = random.randint(1, 8)
        depaLinePost = "n#2#{t}#0 ".format(t=depatime)
    elif src == "c" and dst in stgp3:
        entret = random.randint(1, 8)
    elif src in stgp1 and dst == "e":
        entret = random.randint(16, 20)
        arrLinePost = "p#3#{t}#0 ".format(t=arrtime)
        depaLinePost = "n#2#{t}#0 ".format(t=depatime)
    elif src in stgp1 and dst in stgp3:
        entret = random.randint(1, 8)
        arrLinePost = "p#1#{t}#0 ".format(t=arrtime)
```

```
    else:  # 客机始发
        entret = 0

    return [arrLinePost, entret, depaLinePost]
```

第二部分-生产者

生产者对象

主要部分

- init构造函数部分对于uiautomation自身进行初始化并传入基础参数，如记录日志等
- initsetting设置路路通PC端窗口设置，如当日开行车次、模糊站名等
- 后续的函数均为数据获取，从主窗口点选获取二级窗口中的车次数据并复制到队列内
- del析构函数部分完成日志记录关闭窗口最后销毁对象

待优化部分

1. 增加始发终到查询以及单个车次查询以及筛选列车类型部分
2. ...

In [ ]:
```
class lltskbProcess(object):
    def __init__(self, lltskbroute: str, traincount: int):  # 构造函数，调整设置和

        auto.uiautomation.SetGlobalSearchTimeout(10)
        # set it to False and try again, default is False
        auto.uiautomation.DEBUG_EXIST_DISAPPEAR = True
        # set it to False and try again, default is False
        auto.uiautomation.DEBUG_SEARCH_TIME = True
        subprocess.Popen(
            "start {route}".format(route=lltskbroute), shell=True)
        self.window = auto.WindowControl(searchDepth=1, ClassName='#32770')
        self.window.SetActive()  # 打开程序并激活窗口
        self.rowcount = 18
        self.pagecount = int(traincount/18)+1
        return

    def initsetting(self, querytype=1):  # 对于车型选择（未加入），模糊车站等选择
        # 1为始发站，2为终到站，3为车站查询输入的车站，foundindex不从0开始
        # if querytype==1:
        self.stationquery = self.window.EditControl(
            searchDepth=2, foundIndex=3, AutomationId="1001")
        self.stationquery.Click()  # 点击以激活输入框

        # 返回值为0为未选中，返回值1为选中
        isoperate = auto.CheckBoxControl(
            searchDepth=2, AutomationId="1024")  # 当日不开行
        if isoperate.GetTogglePattern().ToggleState == 1:
            isoperate.Click()  # 勾选以隐藏未开行车次

        # 返回值为0为未选中，返回值1为选中
        isobscure = auto.CheckBoxControl(
            searchDepth=2, AutomationId="1025")  # 模糊车站
        if isobscure.GetTogglePattern().ToggleState == 1:
            isobscure.Click()  # 取消勾选以获得准确单个车站
```

```
        return

    def selectstation(self, station: str):  # 按照车站查询

        self.stationquery.Click()  # 点击以激活输入框
        self.stationquery.SendKeys('{Ctrl}a{Del}', waitTime=0.1)  # 全选清空内容
        self.stationquery.SendKeys(text=station, waitTime=0.1)  # 输入需要查询的车

        self.quetybtn = self.window.ButtonControl(
            searchDepth=1, foundIndex=1, AutomationId="1006")  # 查询按钮
        self.quetybtn.Click(waitTime=0.1)  # 点击查询按钮

        return

    def selecttrain(self, train: str):  # 按照单个车次查询部分，和上面几乎一致

        return

    def generateroute(self, InfoQueueProd, trainCodeQueueProd):

        list1 = self.window.PaneControl(searchDepth=1, foundIndex=1,
                                        AutomationId="1019")  # 获取PaneControl控
        list1.Click(waitTime=0.1)  # 第一次点击激活首页的表格，但是同时会激活单一车次
        list2 = self.window.PaneControl(searchDepth=2, foundIndex=1,
                                        AutomationId="1019")  # 获取PaneControl控
        list1.Click(x=20, y=30, waitTime=0.1)  # 第二次继续点击首页表格重新激活一级
        pgdnbtn = auto.ButtonControl(searchDepth=4,
                                     AutomationId="DownPageButton")  # 翻页按钮
        # for i, j in itertools.product(range(0, pagecount), range(0, rowcount)):
        for j in range(0, self.pagecount):  # 翻页
            for i in range(0, self.rowcount):  # 一页
                # 每一次激活新的二级页面y值向下移动25左右
                list1.Click(x=20, y=30+23*i, waitTime=0.1)

                list2.Click(waitTime=0.1)  # 激活二级页面
                list2.SendKeys('{Ctrl}a{Ctrl}c', waitTime=0.1)  # 全选并复制内容
                result = auto.GetClipboardText()  # 复制到剪切板
                InfoQueueProd.put(result, timeout=1)  # 加入队列
                traincode = list2.GetParentControl().Name
                trainCodeQueueProd.put(traincode, timeout=1)  # 将列车号即二级页面
            try:
                pgdnbtn.Click()  # 下一页按钮，大小会改变，应该需要重新选择?到底了多
            except LookupError as le:
                print(le)
                break
        return

    def __del__(self,):  # 析构函数，记录日志关闭窗口
        auto.Logger.Write('Data acquisition completed.\n',
                          auto.ConsoleColor.Cyan)
        self.window.Disappears(1)  # 确认窗口状态并关闭
        self.window.GetWindowPattern().Close()
        self.window.Exists(1)  # 记录日志

        return
```

生产者工作函数

- 初始化生成者--传入队列等信息，后续多线程主函数调用

- 析构销毁等待完成后自动进行

In [ ]:
```python
def obtaintrain(lltroute: str, tc: int, tarstation: str, InfoQueue: Queue, train
    # 生产者进程函数
    lltpro = lltskbProcess(
        lltskbroute=lltroute, traincount=tc)
    lltpro.initsetting()  # 处理获取时刻表信息
    lltpro.selectstation(station=tarstation)
    lltpro.generateroute(InfoQueue, trainCodeQueue)

    return
```

第三部分-消费者

消费者函数部分

主要内容

依照数据流处理的想法进行编写，所有函数都是只对于一条数据进行处理
由于大部分继承之前按照文本批处理的函数，所以显得比较冗长
主要分为三部分

- initformate 预处理部分重新编写用于对从全局队列中拿到的数据进行预处理-替换异常值
- generateroute 生成径路部分继承自之前批处理部分， 映射对应路径并处理时间部分
- generategame 生成游戏字符串部分同样继承自批处理，生成最终游戏时刻表字符并写入文件
- reproduct 处理生成完毕的文件，主要用于去重(**可选部分，暂未使用**)

待优化部分

1. 优化处理逻辑，更多的适配流处理的思想
2. 优化时间处理部分，尤其是始发终到车
3. ...

In [ ]:
```python
def initStrFormate(station: str, initcodeStr: str, initInfoStr: str):
    # 使用最初得到的字符数据
    traincodeStr = initcodeStr.replace("次", "")  # 字符串样式车次便于最后生成游戏
    if "B" in traincodeStr:  # 一日开行两次的车辆，会导致分割异常
        traincodeStr = traincodeStr.replace("B", "")
    traincodeList = traincodeStr.split(sep="/")  # 列表样式便于下面处理
    if len(traincodeList) == 2:  # 如果变化车次，使用0号位车次替代便于一起处理
        initInfoStr = initInfoStr.replace(
            traincodeList[1], traincodeList[0])  # [1:]

    # 按照车次数字部分分割，因为复制过来车次有首位缺失
    initList = initInfoStr.split(traincodeList[0])
    tarposi = 0
    for i, e in enumerate(initList):
        if station+"\t" in e:  # 目标车站
            tarposi = i  # 获取目标车站位置，暂不考虑环线停靠两次等
            break
    prepareList = [[], [], []]  # 长度为3的空列表
    if tarposi == 1:  # 始发车,0位元素为可能剩下的列车类型字母cdg等
```

```
        prepareList = [[traincodeStr, "src", "00:00", "00:00", "0", "0"],
                       (traincodeStr + initList[1]
                        ).split(sep="\t", maxsplit=6),
                       (traincodeStr+initList[2]).split(sep="\t", maxsplit=6)]
    elif tarposi == len(initList)-1:  # 终到车
        prepareList = [(traincodeStr+initList[tarposi-1]).split(sep="\t", maxspl
                       (traincodeStr + initList[tarposi]
                        ).split(sep="\t", maxsplit=6),
                       [traincodeStr, "dst", "23:59", "23:59", "9999", "0"]]
    else:  # 过路车
        prepareList = [(traincodeStr+initList[tarposi-1]).split(sep="\t", maxspl
                       (traincodeStr + initList[tarposi]
                        ).split(sep="\t", maxsplit=6),
                       (traincodeStr+initList[tarposi+1]).split(sep="\t", maxspl

    for i in range(3):
        while len(prepareList[i]) <= 6:  # 如果最后一列缺失
            prepareList[i].append("0")
        while len(prepareList[i]) > 6:  # 超长的截取
            prepareList[i].pop()
        if prepareList[i][2] == "-- --":  # 替换中心车站时间从-- --到到达时间
            prepareList[i][2] = prepareList[i][3]
        elif prepareList[i][3] == "-- --":  # 替换进场车站离开时间
            prepareList[i][3] = prepareList[i][2]

    return prepareList


def routeStrFormate(prepareList: list[list]):
    # 使用上面初步处理的数据帧并完成进一步处理
    # 时间处理部分
    traindf = pandas.DataFrame(data=prepareList, index=None, columns=[
        "traincode", "station", "arrival", "departure", "totalmiles", "entrance"
    traindf["arrival"] = pandas.to_datetime(
        arg=traindf["arrival"],  format="%H:%M")
    traindf["departure"] = pandas.to_datetime(
        arg=traindf["departure"],  format="%H:%M")
    traindf["stoptime"] = (traindf["departure"] -
                           traindf["arrival"]).dt.total_seconds() / 60
    traindf["stoptime"] = traindf["stoptime"].apply(lambda x: int(x))

    # print(traindf)
    traingpdf = traindf.groupby(
        by="traincode", as_index=False, sort=False).agg(list)

    # dataframe--series--list,三级索引，0和2为前后站索引
    try:
        for k, v in route1.items():
            # 把车站名映射为线路，前后站一定不同所以可以循环判断两次
            if traingpdf["station"][0][0] in v:
                traingpdf["station"][0][0] = k
            elif traingpdf["station"][0][2] in v:
                traingpdf["station"][0][2] = k
    except IndexError:  # 可能会有的超出长度，没有复现成功
        print(traingpdf["traincode"], "出现异常")
        traingpdf["station"][0][0] = traingpdf["station"][0][1]
        traingpdf["station"][0][2] = traingpdf["station"][0][1]

    # arriveroute = traingpdf["station"][0][0]
    # leaveroute = traingpdf["station"][0][2]
```

```
    # 计算理论进场离场时间
    traingpdf["arrival"][0][0] = traingpdf["arrival"][0][1] - \
        datetime.timedelta(
            minutes=(gameStationInfo.get(traingpdf["station"][0][0], gameStation
    traingpdf["departure"][0][2] = traingpdf["departure"][0][1] + \
        datetime.timedelta(
            minutes=(gameStationInfo.get(traingpdf["station"][0][2], gameStation

    return traingpdf
#


def generateGameStr(infodf: pandas.DataFrame, entRule=[]):
    # 最后生成游戏字符串，entrule为股道/检票口在entrance部分的位置范围
    # D5769/D5772 COMMUTER 200 LPPL X1 : 余花联络线南湖东方向#1#19:28:00#0 武汉东#2
    # 车辆信息部分
    trcode = infodf["traincode"][0]
    trtype = trcode[0]
    timfo = species1.get(trtype, ['120', 'LCPPPPPP', "0"])
    trainStr = "{tc} COMMUTER {spe} {mar} X1 : ".format(
        tc=trcode, spe=timfo[0], mar=timfo[1])
    # 进场和立场部分
    ast = infodf["station"][0][0]  # 进场部分
    astref = gameStationInfo.get(ast, gameStationDefault)
    arrStr = "{sta}#{rail}#{clck}#{stay} ".format(
        sta=astref[0], rail=astref[2], clck=infodf["arrival"][0][0], stay=0)
    dst = infodf["station"][0][2]  # 离场部分
    dstref = gameStationInfo.get(dst, gameStationDefault)
    depStr = "{sta}#{rail}#{clck}#{stay} ".format(
        sta=dstref[0], rail=dstref[3], clck=infodf["departure"][0][2], stay=0)
    # 核心车站部分
    cst = infodf["station"][0][1]  # 停站部分
    erl = mapEntrRoute(src=astref[0], dst=dstref[0], arrtime=infodf["arrival"]
                       [0][0], depatime=infodf["departure"][0][2])
    sta1 = infodf["stoptime"][0][1]

    centerStr = "{sta}#{rail}#{clck}#{stay} ".format(
        sta="a", rail=erl[1], clck=infodf["arrival"][0][1], stay=sta1 if sta1 !=
    # 字符串加上线路所
    res = (trainStr+arrStr+erl[0]+centerStr +
           erl[2]+depStr).replace("1900-01-01 ", "")
    # print(res)
    return res


def strReproduct(file1: str, file2: str):
    codelist = []
    infolist = []
    with open(file=file1, mode="r", encoding="utf-8") as f:
        for li in f:
            info1 = li
            code1 = li.split(sep=" ", maxsplit=1)[0]
            if code1 not in codelist:
                codelist.append(code1)
                infolist.append(info1)
            else:
                continue
    f.close()
    infolist.sort()
    with open(file=file2, mode="w", encoding="utf-8") as w:
```

```
        for i in infolist:
            w.writelines(i)
    w.close()

    return
```

消费者工作函数部分

主要内容

使用死循环进行流处理，在所有数据均处理完毕后写入文件

- 使用查空和哨兵值优化性能表现减少占用
  - 如果队列为空则等待5s(消费者远快于生产者，时间还可以在长)，后续等待完成在处理
  - 在接收到magic number后停止退出，完成处理部分
- 数据通过多个函数接续处理，流程函数参考上面，处理完成后加入集合先期去重
- 最后统一写到文件中

In [ ]:
```
def gameStrProcess(station, InfoQueueCons: Queue, trainCodeQueueCons: Queue, out
    resset = set()
    while True:
        if InfoQueueCons.empty() == True:
            print("队列已为空，暂时无元素可初步处理，消费者暂停5s")
            time.sleep(5)
            continue
        initInfo = InfoQueueCons.get(timeout=1)
        if initInfo == "114514": #magic number
            break #处理完成退出
        initcode = trainCodeQueueCons.get(timeout=1)
        # 接续处理
        initpr1 = initStrFormate(
            station=station, initcodeStr=initcode, initInfoStr=initInfo)
        nextpr2 = routeStrFormate(prepareList=initpr1)
        outstr = generateGameStr(nextpr2, [0, 0])
        resset.add(outstr)
    print(resset) #写入文件
    f = open(file=outfile, mode="w", encoding="utf-8")
    for rs in resset:
        f.write(rs+"\n")
        print(rs)
    f.close()
    return
```

第三部分--主函数部分

使用生产者消费者模型建立任务

分别建立生产者和消费者并启动

待优化

1. 多开几个生产者线程，目前不会uiautomation的多线程，会抢鼠标
2. 使用其他数据结构来合并两个公有队列中的字符串，减少锁开销

In [ ]:

```
if __name__ == "__main__":
    InfoQueue1 = multiprocessing.Queue()  # 径路信息队列
    trainCodeQueue1 = multiprocessing.Queue()  # 车次号队列
    # 中继承接队列
    targetstation = "武汉"

    lltskbroute = "lltskb.exe" #最好为程序完整路径，相对路径可能不行
    # 生产者需要设置大致上的列车数目
    p = multiprocessing.Process(
        target=obtaintrain, args=(lltskbroute, 550, targetstation, InfoQueue1, t
    c = multiprocessing.Process(
        target=gameStrProcess, args=(targetstation, InfoQueue1, trainCodeQueue1,

    p.start()  # 启动生产者和消费者进程
    time.sleep(10)  # 启动较慢，等待生产者初始化完成
    c.start()

    p.join()  # 等待生产者进程完成
    InfoQueue1.put("114514")  # magic number,通知消费者所有产品已经生产完毕
    c.join()  # 等待消费者进程完成
```