数据结构课程设计报告

班级：19G212

学号：20211002548

姓名：梁爽

指导老师：李桂玲

日期：2022.07.07

一、离散事件模拟：海关检查点仿真系统

一、课程设计题目与要求

【问题描述】

考虑一个负责检查过境车辆的海关检查点，并开发一个具体的模拟系统。对于该系统，假设有以下基本的考虑因素：
（1）海关的职责是检查过往的车辆，这里只模拟了一个交通检查的方向。
（2）假设车辆以一定的速度到达，就存在一定的随机性，每 a 到 b 分钟就有一个车辆到达。
（3）海关有 k 个检查通道，检查车辆需要 c 到 d 分钟。
（4）到达的车辆在一条专线上排队等候。一旦检查通道畅通，排队的第一辆车辆将进入该通道进行检查。如果车辆到达一个空车道，没有等待车辆，它立即进入车道并开始检查。
（5）所期望的数据包括车辆的平均等待时间和通过检查站的平均时间。

【基本要求】

该系统需要模拟在一个海关检查点的检查过程，并输出一系列事件，以及车辆的平均排队时间和平均过境时间。

【扩展要求】

请修改海关检查点模拟系统，使每个检查通道使用一个等待队列的管理策略。对该新策略进行了仿真，并将仿真结果与共享等待队列的策略进行了比较。

二、需求分析

2.1 输入

用户由图形交互界面的下拉日期选择器输入精确的年、月、日、时、分、秒的模拟开始时间和结束时间，并输入 a, b, c, d, k 的值。

2.2 输出

按时间顺序依次输出模拟的当前时刻与对应时刻发生的事件，在最后输出车辆的平均等待时间和通过检查站的平均时间。

2.3 功能

模拟海关检查，根据输入的时间与参数 a,b,c,d,k 的值模拟检查车辆到达、检查以及离开海关，并计算车辆的平均等待时间和通过检查站的平均时间。

三、设计

3.1 设计思想

3.1.1 数据结构设计

3.1.1.1 存储结构设计

Python 内置库 Datetime.datetime,用于存储与计算精确时刻。

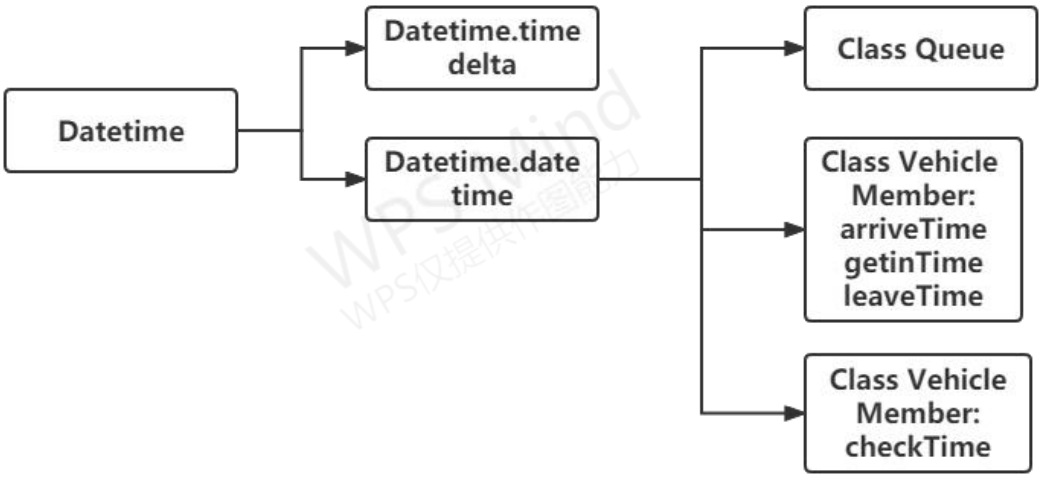
Python 内置库 Datetime.timedelta,用于存储与计算精确时间间隔。

自定义类 Queue，实现循环队列，用于模拟海关等待检查的队列。

自定义类 Vehicle，模拟等待检查的车辆，数据成员为：number(车辆编号，int 类型)、arriveTime（到达时间）、checkTime（检查所需时间间隔）、getinTime（开始检查的时间）、leaveTime（离开时间）。

Class Vehicle		
数据成员	成员类型	意义
number	int	车辆编号
arriveTime	Datetime.datetime	到达时间
checkTime	Datetime.timedelta	检查所需时间间隔
getinTime	Datetime.datetime	开始检查的时间
leaveTime	Datetime.datetime	离开时间

3.1.1.2 逻辑结构设计



3.1.2 算法设计

（1）预处理每辆车的随机数 Vehicle.arriveTime（datetime.timedelta 类型）、Vehicle.checkTime（datetime.timedelta 类型）与其编号，并将 Vehicle 存入列表中。

（2）对 Vehicle 组成的列表 vehicles[i].arriveTime（datetime.timedelta 类型）求前缀和，得到每辆车的具体到达时刻 vehicles[i].arriveTime（datetime.datetime 类型）。

（3）从开始时间开始，遍历每整数分钟直到结束时间。

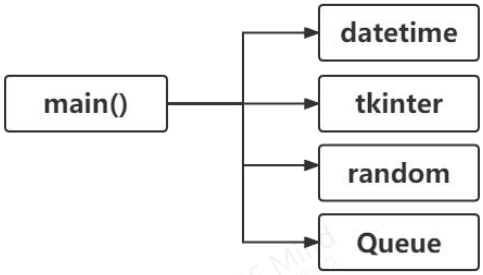
（4）在遍历时间同时，判断 vehicles[i].arriveTime 是否为当前时间，若是，则将 vehicles[i] 加入等待队列。

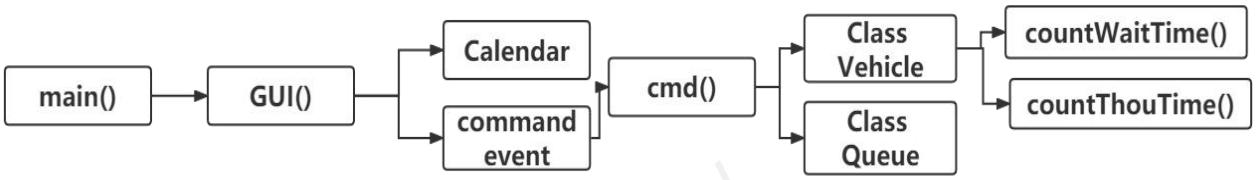
判断检查队列是否为满，若不为满，则将等待队列的头部元素出队，并记录开始检查的时间（vehicle.getinTime）。

判断检查队列的头部元素是否已检查完成，若完成，则将其出队，并记录离开时间（vehicle.leaveTime）。

（5）统计每辆车的平均等待时间和平均通过时间并输出。

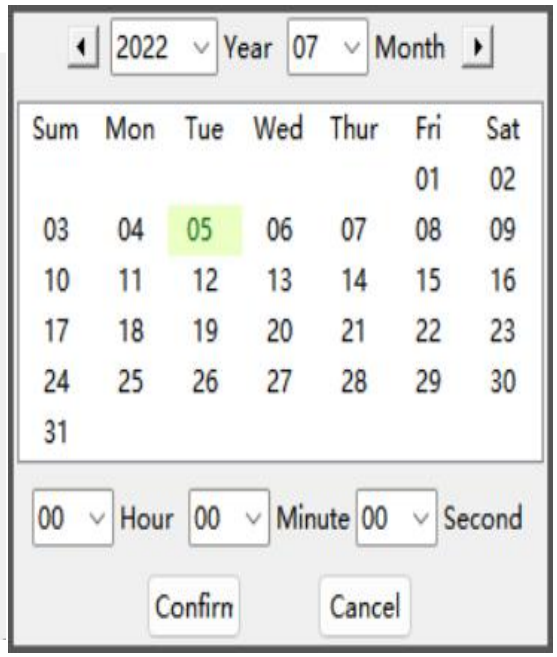
3.2 设计表示





3.3 详细设计

3.3.1 界面设计



3.3.2 核心算法设计

```
while    (currentTime <= endTime):                                  #到结束时间时结束
    if (vehicles[i].arriveTime<=currentTime) then                #判断此时有无车辆到达
        waitQueue.enqueue(vehicles[i])                            #车辆进入等待队列
        Record currentTime as arriveTime                          #记录到达时间
        Print arrive event                                        #打印到达事件
    if (not checkQueue.full()) and (not waitQueue.empty()) #判断检查队列是否为满
        waitQueue.dequeue()                                       #车辆驶出等待队列
        checkQueue.enqueue()                                      #车辆进入检查队列
        Record current time as getinTime                          #记录开始检查的时间
    if (not checkQueue.empty()) then                            #判断检查队列是否为空
        If    checkQueue.peek().getinTime and                   #判断此时是否有车辆离开
              checkQueue.peek().checkTime>=currentTime then
              checkQueue.dequeue()                                #车辆离开检查队列
              Record currentTime as   leaveTime                   #记录离开时间
              Print leave event                                   #打印车辆离开事件
    currentTime+=1 min                                            #模拟下一分钟
```

四、调试分析

4.1 遇到的问题与解决措施

4.1.1 遇到的问题

（1）图形界面交互输入日期。
（2）图形界面难以往核心算法内传递数据。
（3）将获取的时刻字符串转化为 datetime 对象。

4.1.2 解决措施

（1）自定义下拉日历空间 Calendar 类。
（2）将核心算法封装成一个函数，用按钮事件触发。
（3）使用 format 格式控制符或正则表达式。

4.2 时空复杂度分析

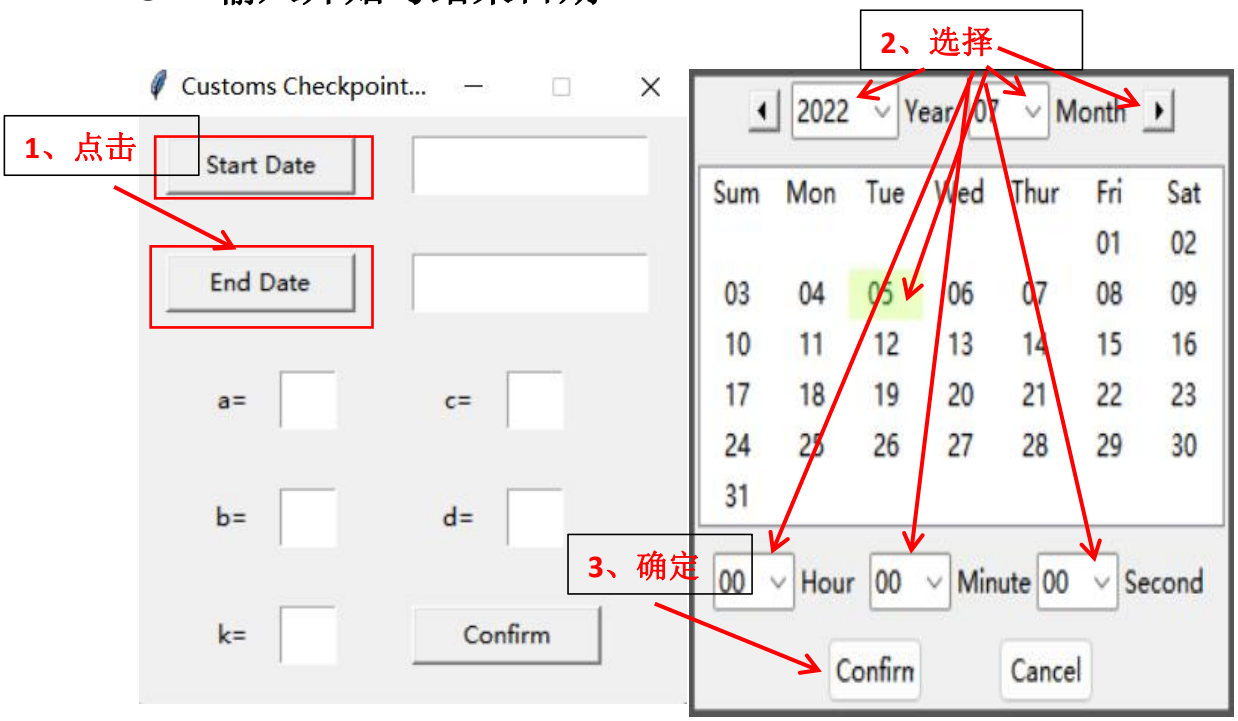
时间复杂度：O(N)
空间复杂度：O(1)

4.3 改进算法

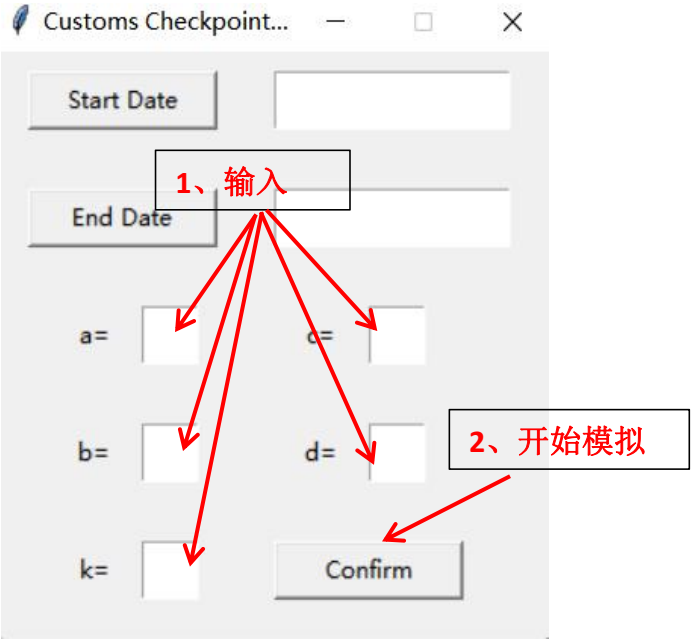
自定义海关检查通道类，使用阻塞队列等等。

五、用户手册

5.1 输入开始与结束日期



5.2 输入其他参数



六、测试数据以及测试结果

6.1 测试用例

参数	值	参数	值
开始时间	2022-07-05 00:00:00	c	10
结束时间	2022-07-05 06:00:00	d	3
a	5	k	5
b	3		

6.2 输出结果

2022-07-05 00:05:00

Event: Vehicle 1 arrive at the custom

2022-07-05 00:05:00

Event: Vehicle 0 leave the custom

2022-07-05 00:15:00

Event: Vehicle 2 arrive at the custom

2022-07-05 00:15:00

Event: Vehicle 1 leave the custom

2022-07-05 00:21:00

Event: Vehicle 3 arrive at the custom

2022-07-05 00:21:00

Event: Vehicle 2 leave the custom

2022-07-05 00:30:00

Event: Vehicle 4 arrive at the custom

2022-07-05 00:30:00

Event: Vehicle 3 leave the custom

2022-07-05 00:39:00
Event: Vehicle 5 arrive at the custom
2022-07-05 00:39:00
Event: Vehicle 4 leave the custom
2022-07-05 00:45:00
Event: Vehicle 6 arrive at the custom
2022-07-05 00:45:00
Event: Vehicle 5 leave the custom
2022-07-05 00:51:00
Event: Vehicle 7 arrive at the custom
2022-07-05 00:51:00
Event: Vehicle 6 leave the custom
2022-07-05 00:59:00
Event: Vehicle 8 arrive at the custom
2022-07-05 00:59:00
Event: Vehicle 7 leave the custom
2022-07-05 01:06:00
Event: Vehicle 9 arrive at the custom
2022-07-05 01:06:00
Event: Vehicle 8 leave the custom
2022-07-05 01:16:00
Event: Vehicle 10 arrive at the custom
2022-07-05 01:16:00
Event: Vehicle 9 leave the custom
2022-07-05 01:23:00
Event: Vehicle 11 arrive at the custom
2022-07-05 01:23:00
Event: Vehicle 10 leave the custom
2022-07-05 01:31:00
Event: Vehicle 12 arrive at the custom
2022-07-05 01:31:00
Event: Vehicle 11 leave the custom
2022-07-05 01:41:00
Event: Vehicle 13 arrive at the custom
2022-07-05 01:41:00
Event: Vehicle 12 leave the custom
2022-07-05 01:50:00
Event: Vehicle 14 arrive at the custom
2022-07-05 01:50:00
Event: Vehicle 13 leave the custom
2022-07-05 01:56:00
Event: Vehicle 15 arrive at the custom
2022-07-05 01:56:00
Event: Vehicle 14 leave the custom
2022-07-05 02:01:00
Event: Vehicle 16 arrive at the custom
2022-07-05 02:01:00
Event: Vehicle 15 leave the custom
2022-07-05 02:10:00
Event: Vehicle 17 arrive at the custom
2022-07-05 02:10:00
Event: Vehicle 16 leave the custom
2022-07-05 02:18:00
Event: Vehicle 18 arrive at the custom
2022-07-05 02:18:00
Event: Vehicle 17 leave the custom
2022-07-05 02:26:00
Event: Vehicle 19 arrive at the custom
2022-07-05 02:26:00
Event: Vehicle 18 leave the custom
2022-07-05 02:31:00
Event: Vehicle 20 arrive at the custom
2022-07-05 02:31:00
Event: Vehicle 19 leave the custom
2022-07-05 02:37:00
Event: Vehicle 21 arrive at the custom
2022-07-05 02:37:00
Event: Vehicle 20 leave the custom
2022-07-05 02:42:00
Event: Vehicle 22 arrive at the custom
2022-07-05 02:42:00
Event: Vehicle 21 leave the custom
2022-07-05 02:50:00
Event: Vehicle 23 arrive at the custom
2022-07-05 02:50:00
Event: Vehicle 22 leave the custom
2022-07-05 02:56:00
Event: Vehicle 24 arrive at the custom
2022-07-05 02:56:00
Event: Vehicle 23 leave the custom
2022-07-05 03:06:00
Event: Vehicle 25 arrive at the custom
2022-07-05 03:06:00
Event: Vehicle 24 leave the custom
2022-07-05 03:15:00
Event: Vehicle 26 arrive at the custom
2022-07-05 03:15:00
Event: Vehicle 25 leave the custom

2022-07-05 03:23:00
Event: Vehicle 27 arrive at the custom
2022-07-05 03:23:00
Event: Vehicle 26 leave the custom
2022-07-05 03:28:00
Event: Vehicle 28 arrive at the custom
2022-07-05 03:28:00
Event: Vehicle 27 leave the custom
2022-07-05 03:34:00
Event: Vehicle 29 arrive at the custom
2022-07-05 03:34:00
Event: Vehicle 28 leave the custom
2022-07-05 03:43:00
Event: Vehicle 30 arrive at the custom
2022-07-05 03:43:00
Event: Vehicle 29 leave the custom
2022-07-05 03:50:00
Event: Vehicle 31 arrive at the custom
2022-07-05 03:50:00
Event: Vehicle 30 leave the custom
2022-07-05 03:56:00
Event: Vehicle 32 arrive at the custom
2022-07-05 03:56:00
Event: Vehicle 31 leave the custom
2022-07-05 04:04:00
Event: Vehicle 33 arrive at the custom
2022-07-05 04:04:00
Event: Vehicle 32 leave the custom
2022-07-05 04:10:00
Event: Vehicle 34 arrive at the custom
2022-07-05 04:10:00
Event: Vehicle 33 leave the custom
2022-07-05 04:20:00
Event: Vehicle 35 arrive at the custom
2022-07-05 04:20:00
Event: Vehicle 34 leave the custom
2022-07-05 04:29:00
Event: Vehicle 36 arrive at the custom
2022-07-05 04:29:00
Event: Vehicle 35 leave the custom
2022-07-05 04:37:00
Event: Vehicle 37 arrive at the custom
2022-07-05 04:37:00
Event: Vehicle 36 leave the custom
2022-07-05 04:44:00
Event: Vehicle 38 arrive at the custom
2022-07-05 04:44:00
Event: Vehicle 37 leave the custom
2022-07-05 04:54:00
Event: Vehicle 39 arrive at the custom
2022-07-05 04:54:00
Event: Vehicle 38 leave the custom
2022-07-05 05:01:00
Event: Vehicle 40 arrive at the custom
2022-07-05 05:01:00
Event: Vehicle 39 leave the custom
2022-07-05 05:09:00
Event: Vehicle 41 arrive at the custom
2022-07-05 05:09:00
Event: Vehicle 40 leave the custom
2022-07-05 05:19:00
Event: Vehicle 42 arrive at the custom
2022-07-05 05:19:00
Event: Vehicle 41 leave the custom
2022-07-05 05:29:00
Event: Vehicle 43 arrive at the custom
2022-07-05 05:29:00
Event: Vehicle 42 leave the custom
2022-07-05 05:36:00
Event: Vehicle 44 arrive at the custom
2022-07-05 05:36:00
Event: Vehicle 43 leave the custom
2022-07-05 05:43:00
Event: Vehicle 45 arrive at the custom
2022-07-05 05:43:00
Event: Vehicle 44 leave the custom
2022-07-05 05:52:00
Event: Vehicle 46 arrive at the custom
2022-07-05 05:52:00
Event: Vehicle 45 leave the custom
2022-07-05 06:00:00
Event: Vehicle 47 arrive at the custom
2022-07-05 06:00:00
Event: Vehicle 46 leave the custom
Average Wait Time: 0:10:00
Average Through Time: 0:07:00

七、源程序清单

7.1 时间控件 Calendar.py

```
import datetime
import calendar
import tkinter as tk
from tkinter import StringVar, ttk
import tkinter.font as tkFont
datetime = calendar.datetime.datetime
timedelta = calendar.datetime.timedelta
class Calendar:
    def __init__(self, point = None):
        # self.root = root
        self.master = tk.Toplevel()
        self.master.withdraw()
        self.master.attributes('-topmost' ,True)
        fwday = calendar.SUNDAY
        year = datetime.now().year
        month = datetime.now().month
        locale = None
        sel_bg = '#ecffc4'
        sel_fg = '#05640e'
        self._date = datetime(year, month, 1) #每月第一日
        self._selection = None #设置为未选中日期
        self.G_Frame = ttk.Frame(self.master)
        self._cal = self.__get_calendar(locale, fwday)
        self.__setup_styles()  # 创建自定义样式
        self.__place_widgets()  # pack/grid 小部件
        self.__config_calendar()  # 调整日历列和安装标记
        # 配置画布和正确的绑定，以选择日期。
        self.__setup_selection(sel_bg, sel_fg)
        # 存储项 ID，用于稍后插入。
        self._items = [self._calendar.insert('', 'end', values='') for _ in range(6)]
        # 在当前空日历中插入日期
        self._update()
        self.G_Frame.pack(expand = 1, fill = 'both')
        self.master.overrideredirect(1)
        self.master.update_idletasks()
        width, height = self.master.winfo_reqwidth(), self.master.winfo_reqheight()
        self.height=height
        if point:
            x, y = point[0], point[1]
        else:
            x, y = (self.master.winfo_screenwidth() - width)/2, (self.master.winfo_screenheight() - height)/2
        self.master.geometry('%dx%d+%d+%d' % (width, height, x, y)) #窗口位置居中
        self.master.after(300, self._main_judge)
        self.master.deiconify()
        self.master.focus_set()
        # self.master.mainloop()
        self.master.wait_window() #这里应该使用 wait_window 挂起窗口，如果使用 mainloop,可能会导致主程序很多错误

    def __get_calendar(self, locale, fwday):
        if locale is None:
            return calendar.TextCalendar(fwday)
        else:
            return calendar.LocaleTextCalendar(fwday, locale)

    def __setitem__(self, item, value):
        if item in ('year', 'month'):
            raise AttributeError("attribute '%s' is not writeable" % item)
        elif item == 'selectbackground':
            self._canvas['background'] = value
        elif item == 'selectforeground':
```

```
self._canvas.itemconfigure(self._canvas.text, item=value)
        else:
            self.G_Frame.__setitem__(self, item, value)
    def __getitem__(self, item):
        if item in ('year', 'month'):
            return getattr(self._date, item)
        elif item == 'selectbackground':
            return self._canvas['background']
        elif item == 'selectforeground':
            return self._canvas.itemcget(self._canvas.text, 'fill')
        else:
            r = ttk.tclobjs_to_py({item: ttk.Frame.__getitem__(self, item)})
            return r[item]
    def __setup_styles(self):
        # 自定义 TTK 风格
        style = ttk.Style(self.master)
        arrow_layout = lambda dir: (
          [('Button.focus', {'children': [('Button.%sarrow' % dir, None)]})]
        )
        style.layout('L.TButton', arrow_layout('left'))
        style.layout('R.TButton', arrow_layout('right'))
    def __place_widgets(self):
        # 标头框架及其小部件
        Input_judgment_num = self.master.register(self.Input_judgment) # 需要将函数包装一下，必要的
        hframe = ttk.Frame(self.G_Frame)
        gframe = ttk.Frame(self.G_Frame)
        bframe = ttk.Frame(self.G_Frame)
        xframe = ttk.Frame(self.G_Frame)
        hframe.pack(in_=self.G_Frame, side='top', pady=5, anchor='center')
        gframe.pack(in_=self.G_Frame, fill=tk.X, pady=5)
        bframe.pack(in_=self.G_Frame, pady=5)
        xframe.pack(in_=self.G_Frame, side='bottom', pady=5)
        lbtn = ttk.Button(hframe, style='L.TButton', command=self._prev_month)
        lbtn.grid(in_=hframe, column=0, row=0, padx=5)
        rbtn = ttk.Button(hframe, style='R.TButton', command=self._next_month)
        rbtn.grid(in_=hframe, column=5, row=0, padx=5)
        #年下拉框
        self.CB_year = ttk.Combobox(hframe, width = 5, values = [str(year) for year in range(datetime.now().year, datetime.now().year-11,-1)], validate = 'key', validatecommand = (Input_judgment_num, '%P'))
        self.CB_year.current(0)
        self.CB_year.grid(in_=hframe, column=1, row=0)
        self.CB_year.bind('<KeyPress>', lambda event:self._update(event, True))
        self.CB_year.bind("<<ComboboxSelected>>", self._update)
        tk.Label(hframe, text = 'Year', justify = 'left').grid(in_=hframe, column=2, row=0, padx=(0,5))
         #月下拉框
        self.CB_month = ttk.Combobox(hframe, width = 3, values = ['%02d' % month for month in range(1,13)], state = 'readonly')

self.CB_month.current(datetime.now().month - 1)
        self.CB_month.grid(in_=hframe, column=3, row=0)

self.CB_month.bind("<<ComboboxSelected>>", self._update)
        tk.Label(hframe, text = 'Month', justify = 'left').grid(in_=hframe, column=4, row=0)
        # 日历部件
        self._calendar = ttk.Treeview(gframe, show='', selectmode='none', height=7)
        self._calendar.pack(expand=1, fill='both', side='bottom', padx=5)
        #时下拉框
        self.CB_hour = ttk.Combobox(bframe, width = 3, values = ['%02d' % hour for hour in range(0,24)],
```

```
validate = 'key', validatecommand = (Input_judgment_num, '%P'))
        self.CB_hour.current(0)
        self.CB_hour.grid(in_=bframe, column=0, row=0)
        self.CB_hour.bind('<KeyPress>', lambda event:self._update(event, True))

self.CB_hour.bind("<<ComboboxSelected>>", self._update)
        tk.Label(bframe, text = 'Hour').grid( in_=bframe,column=1, row=0, padx=(0,5))
        #分下拉框
        self.CB_mins = ttk.Combobox(bframe, width = 3, values = ['%02d' % mins for mins in range(0,60)], validate = 'key', validatecommand = (Input_judgment_num, '%P'))
        self.CB_mins.current(0)
        self.CB_mins.grid(in_=bframe, column=2, row=0)
        self.CB_mins.bind('<KeyPress>', lambda event:self._update(event, True))

self.CB_mins.bind("<<ComboboxSelected>>", self._update)
        tk.Label(bframe, text = 'Minute').grid(column=3, row=0)
        #秒下拉框
        self.CB_seconds = ttk.Combobox(bframe, width = 3, values = ['%02d' % secds for secds in range(0,60)], validate = 'key', validatecommand = (Input_judgment_num, '%P'))
        self.CB_seconds.current(0)
        self.CB_seconds.grid(in_=bframe, column=4, row=0)
        self.CB_seconds.bind('<KeyPress>', lambda event:self._update(event, True))

self.CB_seconds.bind("<<ComboboxSelected>>", self._update)
        tk.Label(bframe, text = 'Second',justify = 'left').grid(column=5, row=0)
        ttk.Button(xframe, text = "Confirm", width = 6, command = lambda: self._exit(True)).grid(row = 1, column = 0, sticky = 'ns', padx = 20)
        ttk.Button(xframe, text = "Cancel", width = 6, command = self._exit).grid(row = 1, column = 4, sticky = 'ne', padx = 20)
        tk.Frame(self.G_Frame, bg = '#565656').place(x = 0, y = 0, relx = 0, rely = 0, relwidth = 1, relheigh = 2/200)
        tk.Frame(self.G_Frame, bg = '#565656').place(x = 0, y = 0, relx = 0, rely = 198/200, relwidth = 1, relheigh = 2/200)
        tk.Frame(self.G_Frame, bg = '#565656').place(x = 0, y = 0, relx = 0, rely = 0, relwidth = 2/200, relheigh = 1)
        tk.Frame(self.G_Frame, bg = '#565656').place(x = 0, y = 0, relx = 198/200, rely = 0, relwidth = 2/200, relheigh = 1)

    def __config_calendar(self):
        # cols = self._cal.formatweekheader(3).split()
        cols = ['Sum','Mon','Tue','Wed','Thur','Fri','Sat']
        self._calendar['columns'] = cols
        self._calendar.tag_configure('header', background='grey90')
        self._calendar.insert('', 'end', values=cols, tag='header')
        # 调整其列宽
        font = tkFont.Font()
        maxwidth = max(font.measure(col) for col in cols)
        for col in cols:
            self._calendar.column(col, width=maxwidth, minwidth=maxwidth,
                anchor='center')

    def __setup_selection(self, sel_bg, sel_fg):
        def __canvas_forget(evt):
            canvas.place_forget()
            self._selection = None

        self._font = tkFont.Font()
```

```
        self._canvas = canvas = tk.Canvas(self._calendar, background=sel_bg, borderwidth=0, highlightthickness=0)
        canvas.text = canvas.create_text(0, 0, fill=sel_fg, anchor='w')
        canvas.bind('<Button-1>', __canvas_forget)
        self._calendar.bind('<Configure>', __canvas_forget)
        self._calendar.bind('<Button-1>', self._pressed)

    def _build_calendar(self):
        year, month = self._date.year, self._date.month
        header = self._cal.formatmonthname(year, month, 0)
        # 更新日历显示的日期
        cal = self._cal.monthdayscalendar(year, month)
        for indx, item in enumerate(self._items):
            week = cal[indx] if indx < len(cal) else []
            fmt_week = [('%02d' % day) if day else '' for day in week]
            self._calendar.item(item, values=fmt_week)

    def _show_select(self, text, bbox):
        x, y, width, height = bbox
        textw = self._font.measure(text)
        canvas = self._canvas
        canvas.configure(width = width, height = height)
        canvas.coords(canvas.text, (width - textw)/2, height / 2 - 1)
        canvas.itemconfigure(canvas.text, text=text)
        canvas.place(in_=self._calendar, x=x, y=y)

    def _pressed(self, evt = None, item = None, column = None, widget = None):
        """在日历的某个地方点击。"""
        if not item:
            x, y, widget = evt.x, evt.y, evt.widget
            item = widget.identify_row(y)
            column = widget.identify_column(x)
        if not column or not item in self._items:
            # 在工作日行中单击或仅在列外单击。
            return
        item_values = widget.item(item)['values']
        if not len(item_values): # 这个月的行是空的。
            return
        text = item_values[int(column[1]) - 1]
        if not text:
            return
        bbox = widget.bbox(item, column)
        if not bbox: # 日历尚不可见
            self.master.after(20, lambda : self._pressed(item = item, column = column, widget = widget))
            return
        text = '%02d' % text  #日
        self._selection = (text, item, column)
        self._show_select(text, bbox)

    def _prev_month(self):
        """更新日历以显示前一个月。"""
        self._canvas.place_forget()
        self._selection = None
        self._date = self._date - timedelta(days=1)
        self._date = datetime(self._date.year, self._date.month, 1)
        self.CB_year.set(self._date.year)
        self.CB_month.set(self._date.month)
        self._update()

    def _next_month(self):
        """更新日历以显示下一个月。"""
        self._canvas.place_forget()
        self._selection = None

        year, month = self._date.year, self._date.month
        self._date = self._date + timedelta(
            days=calendar.monthrange(year, month)[1] + 1)
        self._date = datetime(self._date.year, self._date.month, 1)
```

```
        self.CB_year.set(self._date.year)
        self.CB_month.set(self._date.month)
        self._update()

    def _update(self, event = None, key = None):
        """刷新界面"""
        if key and event.keysym != 'Return': return
        year = int(self.CB_year.get())
        month = int(self.CB_month.get())
        hour = int(self.CB_hour.get())
        mins = int(self.CB_mins.get())
        seconds = int(self.CB_seconds.get())
        if year == 0 or year > 9999: return
        self._canvas.place_forget()
        self._date = datetime(year, month, 1,hour,mins,seconds)
        self._build_calendar() # 重建日历
        if year == datetime.now().year and month == datetime.now().month:
            day = datetime.now().day
            for _item, day_list in enumerate(self._cal.monthdayscalendar(year, month)):
                if day in day_list:
                    item = 'I00' + str(_item + 2)
                    column = '#' + str(day_list.index(day)+1)
                    self.master.after(100, lambda :self._pressed(item = item, column = column, widget = self._calendar))

    def _exit(self, confirm = False):
        if not confirm: self._selection = None
        self.master.destroy()

    def _main_judge(self):
        """判断窗口是否在最顶层"""
        try:
            if self.master.focus_displayof() == None or 'toplevel' not in str(self.master.focus_displayof()): self._exit()
            else: self.master.after(10, self._main_judge)
        except:
            self.master.after(10, self._main_judge)

    def selection(self):
        """返回表示当前选定日期的日期时间。"""

        if not self._selection:
            return None
        year, month = self._date.year, self._date.month
        hour=self._date.hour
        mins = self._date.minute
        seconds = self._date.second
        return str(datetime(year, month, int(self._selection[0]),hour,mins,seconds))

    def Input_judgment(self, content):
        """输入判断"""
        if content.isdigit() or content == "":
            return True
        else:
            return False
```

7.2 循环队列 Queue.py

```
class Queue:
    def __init__(self, limit=10):
        self.data = [None] * limit
        self.head = -1
        self.tail = -1

    def full(self):
        if (self.head+1)%len(self.data)==self.tail:
            return True
        else:
            return False

    def enqueue(self, val): # O(1)
```

```
        if self.empty():
            self.head=0
            self.tail=0
            self.data[0]=val
        else:
            if
(self.head+1)%len(self.data)==self.tail:
                raise RuntimeError()
            else:
                self.head= (self.head + 1) %
len(self.data)
                self.data[self.head] = val

    def peek(self):
        return self.data[self.head]

    def dequeue(self): # O(1)
        if self.empty():
            raise RuntimeError()
        ret = self.data[self.tail]
        self.data[self.tail] = None
        self.tail = (self.tail + 1) % len(self.data)
        if (self.head+1)%len(self.data)==self.tail:
            self.tail=self.head=-1
        return ret

    def resize(self, newsize):
        assert(len(self.data) < newsize)
        newq=Queue(newsize)
        for i in self:
            newq.enqueue(i)
        self.data=newq.data
        self.head=newq.head
        self.tail=newq.tail

    def empty(self):
        if self.head==self.tail==-1:
            return True
        return False

    def __bool__(self):
        return not self.empty()

    def __str__(self):
        if not(self):
            return ''
        return ', '.join(str(x) for x in self)

    def __repr__(self):
        return str(self)

    def __iter__(self):

head=(self.head+len(self.data))%len(self.data)
        tail=(self.tail+len(self.data))%len(self.data)
        i=tail
        while (i!=head):
            yield self.data[i]
            i=(i+1)%len(self.data)
        else:
            yield (self.data[head])
```

7.3 GUI 及主程序 T1_main.py

```
import tkinter as tk
from tkinter import StringVar, ttk
from Calendar import Calendar
from Queue import Queue
from datetime import datetime as dt
from datetime import timedelta as td
import random as rd



class Vehicle():
    def __init__(self, number,arriveTime, checkTime):
        self.number = number
        self.arriveTime = arriveTime
        self.checkTime = checkTime
        self.getinTime = None
        self.leaveTime = None

    def countWaitTime(self):
        return self.leaveTime-self.arriveTime-self.checkTime

    def countThouTime(self):
        return self.leaveTime-self.arriveTime

def GUI():
    def getdate(type):   # 获取选择的日期
        for date in [Calendar().selection()]:
            if date:
                if(type == 'start'):   # 如果是开始按钮，就赋值给开始日期
                    start_date.set(date)
                elif(type == 'end'):
                    end_date.set(date)



    def cmd():
        a=int(entry3.get())
        b=int(entry4.get())
        c=int(entry5.get())
        d=int(entry6.get())
        k=int(entry7.get())
        start=dt.strptime(start_date.get(), "%Y-%m-%d %H:%M:%S")
        end=dt.strptime(end_date.get(), "%Y-%m-%d %H:%M:%S")
        currentTime=start

        vehicles = [Vehicle(0,start+td(minutes=rd.randint(a, b)),

checkTime=td(minutes=rd.randint(c, d)))]
        n = int((end-start).total_seconds()//60//a)
        for i in range(1, n):

vehicles.append(Vehicle(i,vehicles[i-1].arriveTime+td(minutes=rd.randint(a, b)), td(minutes=rd.randint(c, d))))

        waitQueue = Queue(limit=n)
        checkQueue = Queue(limit=k)

        i=0
        l=0
        while (currentTime <= end):
            if vehicles[i].arriveTime<=currentTime:
                waitQueue.enqueue(vehicles[i])

vehicles[i].arriveTime=currentTime
                i+=1
                print(currentTime)

print("Event:","Vehicle",vehicles[i].number,"arrive at the custom")

            if (not checkQueue.full()) and (not waitQueue.empty()):
                tmp=waitQueue.dequeue()
                checkQueue.enqueue(tmp)
                tmp.getinTime = currentTime
            if not checkQueue.empty():
                if checkQueue.peek().getinTime +
```

```
checkQueue.peek().checkTime>=currentTime:
                    tmp=checkQueue.dequeue()
vehicles[i].leaveTime=currentTime
                    l+=1
                    print(currentTime)
print("Event:","Vehicle",tmp.number,"leave the custom")
            currentTime+=td(minutes=1)

        avgeWaitTime=td()
        avgeThouTime=td()
        for j in range(l):
avgeWaitTime+=vehicles[i].countWaitTime()

avgeThouTime+=vehicles[i].countThouTime()

        print("Average Wait Time:",-avgeWaitTime/l)
        print("Average Through Time:",-avgeThouTime/l)



    root = tk.Tk()
    root.title('Customs Checkpoint Simulation System')
    root.geometry('300x300')
    root.resizable(False,False)
    start_date=tk.StringVar()
    end_date=tk.StringVar()
    button1=tk.Button(root,
                      width=15,
                      text='Start Date',
                      command=lambda: getdate('start'))
    button1.place(x=20,
                  y=10,
                  width=100,
                  height=30)
    entry1=tk.Entry(root, textvariable=start_date)
    entry1.place(x=150,
                 y=10,
                 width=125,
                 height=30)
    button2=tk.Button(root, width=15, text='End Date', command=lambda: getdate(
        'end'))
    button2.place(x=20,
                  y=70,
                  width=100,
                  height=30)
    entry2=tk.Entry(root,textvariable=end_date)
    entry2.place(x=150,
                 y=70,
                 width=125,
                 height=30)

    button3=tk.Button(root,text="Confirm",command = cmd)

    entry3=tk.Entry(root,width=5)
    entry4=tk.Entry(root,width=5)
    entry5=tk.Entry(root,width=5)
    entry6=tk.Entry(root,width=5)
    entry7=tk.Entry(root,width=5)

    label1=tk.Label(root,text='a=')
    label2=tk.Label(root,text='b=')
    label3=tk.Label(root,text='c=')
    label4=tk.Label(root,text='d=')
    label5=tk.Label(root,text='k=')

    entry3.place(x=80,
                 y=130,
                 width=30,
                 height=30)
    label1.place(x=40,
                 y=130,
                 width=30,
                 height=30)

    entry4.place(x=80,
                 y=190,
```

```
                width=30,
                height=30)
label2.place(x=40,
                y=190,
                width=30,
                height=30)

entry5.place(x=200,
                y=130,
                width=30,
                height=30)
label3.place(x=160,
                y=130,
                width=30,
                height=30)

entry6.place(x=200,
                y=190,
                width=30,
                height=30)
label4.place(x=160,
                y=190,
                width=30,
                height=30)

        entry7.place(x=80,
                y=250,
                width=30,
                height=30)
        label5.place(x=40,
                y=250,
                width=30,
                height=30)

        button3.place(x=150,
                y=250,
                width=100,
                height=30)
        root.mainloop()

GUI()
```

二、计算命题演算公式的真值

一、课程设计题目与要求

【问题描述】

命题演算公式是指由逻辑变量(其值为 TRUE 或 false)和逻辑运算符∧(AND)、∨(OR)和(NOT)根据某些规则（如含义等操作可以使用∧、∨和表示）组成的公式。公式操作顺序为，∧、∨，括号()可以改变优先级。给定一个命题演算公式和每个变量的值，需要设计一个程序来计算公式的真值。

【基本要求】

（1）使用二叉树来计算公式的真值。 首先，使用堆栈将公式的中缀形式更改为后缀形式。其次，根据后缀形式， 从叶节点构造相应的二叉树。最后，按后序遍历二叉树， 并找到每个子树的值。也就是说，每次到达一个节点时，其子树的值都已为计算出的当 达到根节点时，就得到了公式的真值。

（2）设计了各种不同形式的命题计算公式，并验证了每个命题计算公式的有效性。

（3）逻辑参数的标识符并不局限于单个字母，还可以是任意长度的字母数字字符串。逻辑参数可以在公式中出现多次。

（4）打印二进制树的构造过程，打印公式的后缀形式和二进制树的后序遍历序列。

（5）输入每个变量的值，计算并显示公式的真值，打印二叉树的评估过程。

（6）显示该公式的真值表。

【扩展要求】

请将逻辑运算符替换为算术运算符，并使用二叉树来计算算术表达式。

二、需求分析

2.1 输入

输入命题演算公式。

2.2 功能

（1）检验命题演算公式的合法性。
（2）将命题演算公式转化为后缀表达式。
（3）将后缀表达式转化为表达式树，并打印其构建过程。
（4）利用表达式树计算命题演算公式的真值，并打印其真值表。

2.3 输出

输出命题演算公式的后缀表达式，表达式树的构建过程，命题演算公式的真值表。

三、设计

3.1 设计思想

3.1.1 数据结构设计

（1）逻辑结构设计

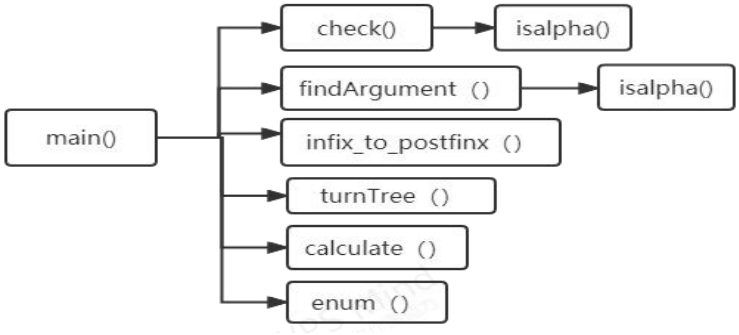
（2）存储结构设计

1、使用字典存储运算符的优先级，便于使用时查字典。
2、使用二叉树存储表达式树。
3、使用栈构建后缀表达式与表达式树。

3.1.2 算法设计

（1）利用栈检查命题公式的合法性。
（2）利用线性查找区分命题变元与运算符。
（3）利用栈将中缀表达式转化为后缀表达式。
（4）利用栈将后缀表达式转化为表达式树。
（5）利用二叉树递归计算命题真值。
（6）利用进制转化生成命题公式的所有解释。

3.2 设计表示



3.3 详细设计

3.3.1 检验命题公式合法性

```
#区分命题变元与真值
def isalpha(word):
    If ('!','∧','∨','(',')',' ') is in word
        return False
    else
        return True

#检查合法性
def check(expr):
    s = Stack()
#检查命题公式开头和结尾的合法性
    if expr begins with ['∧','∨'] or ends with ['∧','∨','!']
        return False
    else:
#列举运算符不合法情况
        if expr[i] in ['∧','∨'] and expr[i+1] in ['∧','∨',')']
            return False
        elif expr[i] in ['!','('] and expr[i+1] in ['∧','∨',')']:
            return False
        elif expr[i]=='(':expr[i+1] in ['∧','∨']:
            return False
        elif expr[i]==')' and expr[i+1]=='!':
            return False
#检查括号合法性
        if expr[i] == '(':
            s.push(expr[i])
        elif expr[i] == ')':
            if s.empty():
                return False
            elif s.pop() != '(':
                return False
    return s.empty()
```

3.3.2 中缀表达式转化为后缀表达式

（1）创建一个栈和一个线性表。

（2）遍历 infix，数组元素为命题变元直接进入线性表；数组元素为运算符和()，先进栈。

a.元素进栈时首先获取栈顶元素优先级，如果该元素的优先级大于等于栈顶元素，栈顶元素弹栈并存入线性表。

b.如果该元素为“（”直接进栈。

c.如果栈内为空或者栈顶元素为“（”则直接进栈。

d.如果元素为“）”栈中元素弹栈并存入线性表直到栈顶元素为“（”。

（4）将栈中剩余的元素依次弹出并放入线性表。

```
def infix_to_postfix(expr):
    ops = Stack()
    postfix = []
    toks = expr.split()
#处理括号
    for c in toks:
        if c == '(':
            ops.push(c)。
        elif c == ')':
            while ops.peek() != '(':
                postfix.append(ops.pop())
            ops.pop()
#处理命题变元
        elif isalpha(c):
            postfix.append(c)
        else:
#按运算符优先级入栈
            while bool(ops) and prec[ops.peek()] >= prec[c]:
                postfix.append(ops.pop())
            ops.push(c)
#将栈中剩余的运算符出栈
    while bool(ops):
        postfix.append(ops.pop())
    return postfix
```

3.3.3 后缀表达式转换为表达式树

（1）创建一个栈，将后缀表达式的每一个符号准备压入栈中。

（2）如果该符号是数字，该数字表示单节点树，压入栈。

（3）如果该符号是二元运算符，从栈中一次 Pop 出两个元素，第一个为右子树，第二个为左子树，然后将新生成的树压入栈。

（4）如果该符号是一元运算符，从栈中一次 Pop 出一个元素为右子树，然后将新生成的树压入栈。

（5）最终栈中只剩一个树节点元素，将其 Pop 出，其为表达式树的根节点。

```
def turnTree(postfix):
    s=Stack()
    for i in postfix:
        if isalpha(i):
            s.push(BTree.Node(i))
        elif i=='!':
            a=s.pop()
            tmp=BTree.Node(i,left=None,right=a)
            BTree.pprint(tmp)
            s.push(tmp)
        elif i in ['∧','∨']:
            a=s.pop()
            b=s.pop()
```

```
tmp=BTree.Node(i,left=b,right=a)
            BTree.pprint(tmp)
            s.push(tmp)
t=BTree()
t.root=s.pop()
return t
```

3.3.4 计算表达式树

（1）从根节点开始遍历表达式树。
（2）若结点值为运算符，递归调用 calculate 函数将此结点作为根节点计算其值。
（3）若结点值为命题变元，返回其真值。

```
def calculate(node):
    if node:
        if node.val not in ('∧','∨','!'):
            return values[node.val]
        elif node.val =='!':
            return not calculate(node.left)
        elif node.val == '∧':
            return calculate(node.left) and calculate(node.right)
        elif node.val == '∨':
            return calculate(node.left) or calculate(node.right)
```

3.3.5 查找命题公式中的所有不重复命题变元

（1）遍历命题公式，若不为运算符则加入命题变元列表中

```
def findArgument(expr):
    arguments=[]
    temp=expr.split()
    for i in temp:
        if i not in ('!','∧','∨','(',')',' '):
            arguments.append(i)
return arguments
```

3.3.6 枚举命题变元的所有真值

（1）遍历区间[0,2^n],n 为命题变元个数。
（2）将 i 转化为二进制，再将其转化为字符串，不足 n 位的补充前导‘0’。
（3）将 str 每位分别转为十进制，对应第 n 个命题变元的真值，并将其添加到真值字典中。
（4）每执行一次循环，将真值字典添加到真值字典列表中。

```
def enum(arguements):
    lst=[]
    for i in range(2**len(arguments)):
        dic={}
        s=str(bin(i))[2::]
        if len(s)<len(arguments):
            s='0'*(len(arguments)-len(s))+s
        s=[int (k) for k in s]
        for j in range(len(arguments)):
            dic[arguments[j]]=s[j]
        lst.append(dic)
return lst
```

3.3.7 二叉树的可视化

```
import networkx as nx
import matplotlib.pyplot as plt
#递归遍历二叉树，计算每个结点的坐标
def create_graph(G, node, pos={}, x=0, y=0, layer=1):
    pos[node.val] = (x, y)
    if node.left:
        G.add_edge(node.val, node.left.val)
        l_x, l_y = x - 1 / 2 ** layer, y - 1
        l_layer = layer + 1
        create_graph(G, node.left, x=l_x, y=l_y, pos=pos, layer=l_layer)
    if node.right:
        G.add_edge(node.val, node.right.val)
        r_x, r_y = x + 1 / 2 ** layer, y - 1
        r_layer = layer + 1
        create_graph(G, node.right, x=r_x, y=r_y, pos=pos, layer=r_layer)
    return (G, pos)

def draw(node):    # 以某个节点为根画图
    graph = nx.DiGraph()
    graph, pos = create_graph(graph, node)
    fig, ax = plt.subplots(figsize=(8, 10))
    nx.draw_networkx(graph, pos, ax=ax, node_size=1000)
    plt.show()
```

3.3.8 拓展要求——计算算数表达式的值

```
from Stack import Stack
import BTreeVisualization as BTV
from BTree import BTree

prec = {'*': 2,'/': 2, '+': 1,'-':1,'(': 0, ')': 0}

def infix_to_postfix(expr):
    ops = Stack()
    postfix = []
    toks = expr.split()

    for c in toks:
        if c == '(':
            ops.push(c)
        elif c == ')':
            while ops.peek() != '(':
                postfix.append(ops.pop())
            ops.pop()
        elif c.isdigit():
            postfix.append(c)
        else:
            while bool(ops) and prec[ops.peek()] >= prec[c]:
                postfix.append(ops.pop())
            ops.push(c)
    while bool(ops):
        postfix.append(ops.pop())
    return postfix

def turnTree(postfix):
    s=Stack()
    for i in postfix:
        if i.isdigit():
            s.push(BTree.Node(i))
        elif i in ['+','-','/','*']:
            a=s.pop()
            b=s.pop()

tmp=BTree.Node(i,left=b,right=a)
            BTree.pprint(tmp)
            s.push(tmp)
```

```
    t=BTree()
    t.root=s.pop()
    return t

def calculate(node):
    if node:
        if node.val not in ('+','-','*','/'):
            return int(node.val)
        elif node.val =='+':
            return calculate(node.left) + calculate(node.right)
        elif node.val == '-':
            return calculate(node.left) - calculate(node.right)
        elif node.val == '*':
            return calculate(node.left) * calculate(node.right)
        elif node.val == '/':
            return calculate(node.left) / calculate(node.right)

a=input()
a=infix_to_postfix(a)
print(a)
a=turnTree(a)
print(calculate(a.root))
```

四、调试分析

4.1 遇到的问题与解决办法

遇到的问题：构建表达式树时，二元运算符命题变元的前后顺序倒置。

解决办法：借助栈构建表达式树时，将先 pop 的命题变元作为右结点，后 pop 的命题变元作为左节点。

4.2 时空复杂度

时间复杂度：O(N)

空间复杂度：O(N)

五、用户手册

5.1 输入

输入一个命题公式，各命题变元与运算符之间用空格隔开，若公式不合法，则依据提示重新输入。

5.2 输出

依次输出后缀表达式（二叉树的后序遍历），表达式树的构建过程，表达式树，表达式树的真值表。

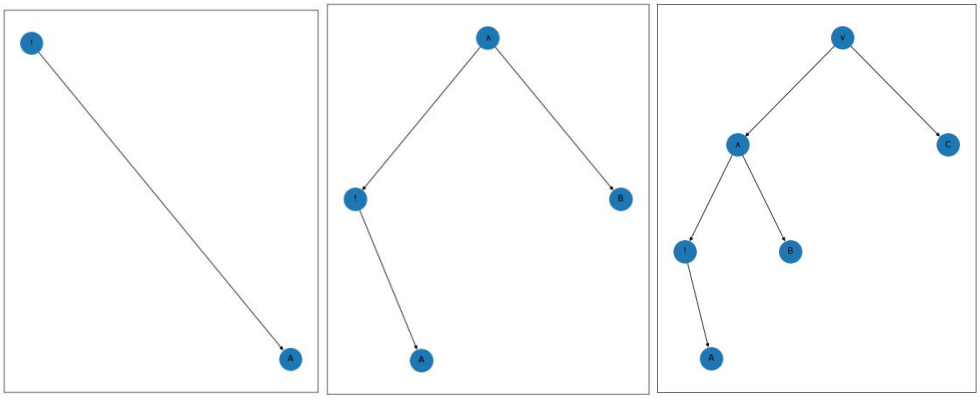
六、测试数据及测试结果

测试用例一

测试输入	∧ (! A ∧ B) ∨ C
测试目的	检验检查公式合法性的功能
正确输出	Retry!
实际输出	Retry!
错误原因	
当前状态	通过

测试输入 ∧ (! A ∧ B) ∨ C
测试目的 检验检查公式合法性的功能
当前状态 通过

测试用例二

测试输入	(! A ∧ B) ∨ C
测试目的	检查中缀表达式转后缀表达式的功能，检查建立表达式树并可视化的功能
正确输出	A ! B ∧ C ∨
实际输出	A ! B ∧ C ∨ 
错误原因	
当前状态	通过

测试用例三

测试输入	(! A ∧ B) ∨ A ∧ C
测试目的	检验判断命题变元个数，打印真值表的功能
正确输出	{'A': 0, 'B': 0, 'C': 0} 0 {'A': 1, 'B': 0, 'C': 0} 0 {'A': 0, 'B': 1, 'C': 0} 0 {'A': 0, 'B': 0, 'C': 1} 0 {'A': 1, 'B': 0, 'C': 1} 1 {'A': 0, 'B': 1, 'C': 1} 1

	{'A': 1, 'B': 1, 'C': 0} 0 {'A': 0, 'B': 0, 'C': 0} 0 {'A': 1, 'B': 0, 'C': 0} 0 {'A': 0, 'B': 1, 'C': 0} 0 {'A': 1, 'B': 1, 'C': 0} 0	{'A': 1, 'B': 1, 'C': 1} 1 {'A': 0, 'B': 0, 'C': 1} 0 {'A': 1, 'B': 0, 'C': 1} 1 {'A': 0, 'B': 1, 'C': 1} 1 {'A': 1, 'B': 1, 'C': 1} 1
实际输出	{'A': 0, 'B': 0, 'C': 0} 0 {'A': 1, 'B': 0, 'C': 0} 0 {'A': 0, 'B': 1, 'C': 0} 0 {'A': 1, 'B': 1, 'C': 0} 0 {'A': 0, 'B': 0, 'C': 0} 0 {'A': 1, 'B': 0, 'C': 0} 0 {'A': 0, 'B': 1, 'C': 0} 0 {'A': 1, 'B': 1, 'C': 0} 0	{'A': 0, 'B': 0, 'C': 1} 0 {'A': 1, 'B': 0, 'C': 1} 1 {'A': 0, 'B': 1, 'C': 1} 1 {'A': 1, 'B': 1, 'C': 1} 1 {'A': 0, 'B': 0, 'C': 1} 0 {'A': 1, 'B': 0, 'C': 1} 1 {'A': 0, 'B': 1, 'C': 1} 1 {'A': 1, 'B': 1, 'C': 1} 1
错误原因		
当前状态	通过	

七、源程序清单

7.1 简单栈的实现 Stack.py

```
class Stack:
    def __init__(self):
        self.data = []

    def push(self, val):
        self.data.append(val)

    def pop(self):
        assert not self.empty()
        ret = self.data[-1]
        del self.data[-1]
        return ret

    def peek(self):
        assert not self.empty()
        return self.data[-1]

    def empty(self):
        return len(self.data) == 0

    def __bool__(self):
        return not self.empty()
```

7.2 二叉树的实现 BTree.py

```
import networkx as nx
import matplotlib.pyplot as plt


class BTree:
    class Node:
        def __init__(self, val, left=None, right=None):
            self.val = val
            self.left = left
            self.right = right

    def __iter__(self):
        def iter_rec(n):
            if n:
                yield from iter_rec(n.left)
```

```
                yield from iter_rec(n.right)
                yield n
        return iter_rec(self.root)


    def __init__(self):
        self.root = None

    def height(node):
        def height_rec(t):
            if not t:
                return 0
            else:
                return 1 + max(height_rec(t.left), height_rec(t.right))
        return height_rec(node)

    def pprint(node,width=128):
        height = BTree.height(node)
        nodes    = [(node, 0)]
        prev_level = 0
        repr_str = ''
        while nodes:
            n,level = nodes.pop(0)
            if prev_level != level:
                prev_level = level
                repr_str += '\n'
            if not n:
                if level < height-1:
                    nodes.extend([(None, level+1), (None, level+1)])
                repr_str += '{val:^{width}}'.format(val='-', width=width//2**level)
            elif n:
                if n.left or level < height-1:
                    nodes.append((n.left, level+1))
                if n.right or level < height-1:
                    nodes.append((n.right, level+1))
                repr_str += '{val:^{width}}'.format(val=n.val, width=width//2**level)
        print(repr_str)
        print('-'*128)
```

7.3 二叉树可视化 BtreeVisualization.py

```
import networkx as nx
import matplotlib.pyplot as plt

def create_graph(G, node, pos={}, x=0, y=0, layer=1):
    pos[node.val] = (x, y)
    if node.left:
        G.add_edge(node.val, node.left.val)
        l_x, l_y = x - 1 / 2 ** layer, y - 1
        l_layer = layer + 1
        create_graph(G, node.left, x=l_x, y=l_y, pos=pos, layer=l_layer)
    if node.right:
        G.add_edge(node.val, node.right.val)
        r_x, r_y = x + 1 / 2 ** layer, y - 1
        r_layer = layer + 1
        create_graph(G, node.right, x=r_x, y=r_y, pos=pos, layer=r_layer)
    return (G, pos)

def draw(node):    # 以某个节点为根画图
    graph = nx.DiGraph()
    graph, pos = create_graph(graph, node)
    fig, ax = plt.subplots(figsize=(8, 10))
    nx.draw_networkx(graph, pos, ax=ax, node_size=1000)
    plt.show()
```

7.4 主程序 T2_main.py

```
from Stack import Stack
import BTreeVisualization as BTV
from BTree import BTree

def isalpha(word):
    for i in ('!','∧','∨','(',')',' '):
        if i in word:
            return False
    return True

def findArgument(expr):
    arguments=[]
    temp=expr.split()
    for i in temp:
        if i not in    ('!','∧','∨','(',')',' '):
            arguments.append(i)
    return arguments

def check(expr):
    s = Stack()
    expr=expr
    if expr[0] in ['∧','∨']:
        return False
    elif (expr[-1] in ['∧','∨','!']):
        return False
    else:
        for i in range(len(expr)):
            if i<len(expr)-1:
                if expr[i] in ['∧','∨']:
                    if  expr[i+1]  in  [' ∧ ','
∨',')']:
                        return False
                elif expr[i]=='!':
                    if  expr[i+1]  in  [' ∧ ','
∨',')']:
                        return False
                elif expr[i]=='(':
                    if  expr[i+1]  in  [' ∧ ','
∨']:
                        return False
                elif expr[i]==')':
                    if expr[i+1]=='!':
                        return False

            if expr[i] == '(':
                s.push(expr[i])
            elif expr[i] == ')':
                if s.empty():
                    return False
                elif s.pop() != '(':
                    return False

    return s.empty()

prec = {'!': 2,'∧': 1, '∨': 1,'(': 0, ')': 0}

def infix_to_postfix(expr):
    ops = Stack()
    postfix = []
    toks = expr.split()

    for c in toks:
        if c == '(':
            ops.push(c)
        elif c == ')':
            while ops.peek() != '(':
                postfix.append(ops.pop())
            ops.pop()
        elif isalpha(c):
            postfix.append(c)
        else:
            while        bool(ops)        and
prec[ops.peek()] >= prec[c]:
                postfix.append(ops.pop())
            ops.push(c)
    while bool(ops):
        postfix.append(ops.pop())
    return postfix

def turnTree(postfix):
    s=Stack()
    for i in postfix:
```

```
            if isalpha(i):
                s.push(BTree.Node(i))
            elif i=='!':
                a=s.pop()

tmp=BTree.Node(i,left=None,right=a)
                BTree.pprint(tmp)
                s.push(tmp)
            elif i in ['∧','∨']:
                a=s.pop()
                b=s.pop()

tmp=BTree.Node(i,left=b,right=a)
                BTree.pprint(tmp)
                s.push(tmp)

    t=BTree()
    t.root=s.pop()
    return t

def calculate(node):
    if node:
        if node.val not in ('∧','∨','!'):
            return values[node.val]
        elif node.val =='!':
            return not calculate(node.left)
        elif node.val == '∧':
            return  calculate(node.left)  and
calculate(node.right)
        elif node.val == '∨':
            return   calculate(node.left)   or
calculate(node.right)
def enum(arguements):
    lst=[]
    for i in range(2**len(arguments)):
        dic={}
        s=str(bin(i))[2::]
        if len(s)<len(arguments):
            s='0'*(len(arguments)-len(s))+s
        s=[int (k) for k in s]
        for j in range(len(arguments)):
            dic[arguments[j]]=s[j]
        lst.append(dic)
    return lst

print("请输入命题公式 示例:( ! A ∧ B ) ∨
C")
expr=input()
values=None
legal=check(expr)
while not legal:
    print("Retry!")
    expr=input()
    legal=check(expr)
else:
    arguments=findArgument(expr)
    expr1=infix_to_postfix(expr)
    t=turnTree(expr1)
    explain=enum(arguments)
    BTree.pprint(t.root)

    for i in range(2**len(arguments)):
        values=explain[i]
        print(explain[i],calculate(t.root))
```

7.5 拓展要求主程序 T2_ExtendedRequirements.py

```
from Stack import Stack
import BTreeVisualization as BTV
from BTree import BTree

prec = {'*': 2,'/': 2, '+': 1,'-':1,'(': 0, ')': 0}

def infix_to_postfix(expr):
    ops = Stack()
    postfix = []
    toks = expr.split()

    for c in toks:
        if c == '(':
            ops.push(c)
```

```
        elif c == ')':
            while ops.peek() != '(':
                postfix.append(ops.pop())
            ops.pop()
        elif c.isdigit():
            postfix.append(c)
        else:
            while bool(ops) and prec[ops.peek()] >= prec[c]:
                postfix.append(ops.pop())
            ops.push(c)
    while bool(ops):
        postfix.append(ops.pop())
    return postfix

def turnTree(postfix):
    s=Stack()
    for i in postfix:
        if i.isdigit():
            s.push(BTree.Node(i))
        elif i in ['+','-','/','*']:
            a=s.pop()
            b=s.pop()

tmp=BTree.Node(i,left=b,right=a)
            BTree.pprint(tmp)
            s.push(tmp)
    t=BTree()
    t.root=s.pop()
    return t

def calculate(node):
    if node:
        if node.val not in ('+','-','*','/'):
            return int(node.val)
        elif node.val =='+':
            return calculate(node.left) + calculate(node.right)
        elif node.val == '-':
            return calculate(node.left) - calculate(node.right)
        elif node.val == '*':
            return calculate(node.left) * calculate(node.right)
        elif node.val == '/':
            return calculate(node.left) / calculate(node.right)

a=input()
a=infix_to_postfix(a)
print(a)
a=turnTree(a)
print(calculate(a.root))
```