Justin Minsk

DATA 520

Project Report

Store Router

**Introduction**

My application routes the smallest distance to travel in a store using a grocery list created by the user, thus taking the shortest amount of time in the store. I created this program since other applications utilize list creation and shared lists to help with shopping, but there are currently no applications that look at a grid of the store and route shopping. As a ‘list shopper’ I want to be able to get in and out of a store as fast as possible and this application would help in achieving that goal. I used basic python programing with the inclusion of the Tkinter package to create an application that routes, but also is easy to use with a simple GUI.

**History**

The start of my project was to create a sample store to use as an example. I learned that a working grid of the store was not publicly accessible which led to me create a store with a grid in python. I did this by combining three lists into a dictionary with the key being the item and the call being a tuple with the coordinates. The next task was to create a traveling salesman algorithm, which would prove to be the greater of the challenges. First was to figure out how to calculate distance, which I learned, in a store containing shelves, that Manhattan distance or block distance was the way to find the true walking distance from point A to point B. Before I had this worked out, I was using the standard distance formula as a place holder until I had time to add Manhattan distance instead.

The rest of the traveling salesman algorithm was figuring out how to make it find the shortest distance going through the store. This meant taking the items and making sure that all items where routed to once and only once. I then decided to move forward with the simplest traveling salesmen solution, which would be the greedy algorithm. While not always the best at finding the shortest, it was still faster then routing by hand. This meant that after arriving at a point the algorithm I would check to see what the next closest item was. I used a series of loops and if statements to create the logic that would create my greedy traveling salesman algorithm. The last problem to solve for the traveling salesman algorithm was to create a result list which I used Pythons’ lists to create a list of items in order and a list of grid coordinates.

The last thing to make was the GUI, which I started in Kivey, but ended up doing in Tkinter packages. I liked the look of the dropdown boxes in Kivey but after some play I found it too time consuming to create a GUI that would allow you to pick up to 33 items using drop boxes. That is when I moved to Tkinter checkboxes. After setting the checkboxes up, one for each of my 33 items, I created a button to route the items and open a new screen with the results. I also created a quit button for easy escape from the application. On my result window I used Tkinters formatting to create a table with the name of items in order from top to bottom and the coordinates next to the names. This created a clean and easy to use application. I finished the process by adding some labels to add instructions to explain how to use the application.

I tested the code by creating a matrix with all of the distances using this code:

def grid\_name():  
 *"""() -> 2d array  
 Create a store array to use in the store router program. This creates the item name.  
 """* store\_item = ['entrance', 'cheese', 'milk', 'yogurt', 'chicken', 'beef', 'pork', 'tv dinners', 'ice cream', 'waffles',  
 'cereal', 'coffee', 'tea', 'bread', 'cake', 'crackers', 'cookies', 'water', 'soda', 'juice',  
 'vegetables', 'fruit', 'salads', 'nuts', 'jam', 'peanut butter', 'paper towels', 'tooth paste',  
 'laundry detergent']  
 return store\_item  
  
  
def grid\_measurements():  
 *"""() -> 2d array  
 Create a store array to use in store router program. This creates a list of measurements.  
 """* store\_mesurements = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 3, 3, 3, 3, 3, 3, 3, 3, 3, 6, 6, 6, 6, 6, 6, 6, 6, 6]  
 return store\_mesurements  
  
  
def grid\_id(grid\_measurements):  
 *"""(array) -> 2d array  
 Create a store array to use in store router program. This creates a list of ids.  
 """* store\_id =[0]  
 count = 1  
 for mesurement in grid\_measurements:  
 if count <= 9:  
 store\_id.append(count)  
 count += 1  
 else:  
 count = 1  
 store\_id.append(count)  
 return store\_id  
  
  
def grid\_create(grid\_name, grid\_measurements, grid\_id):  
 *"""(arrays) -> dict  
 Create the grid for the store router app.  
 """* grid\_zip = zip(grid\_measurements, grid\_id)  
 grid\_list = tuple(grid\_zip)  
 grid\_zip = zip(grid\_name, grid\_list)  
 grid\_dict = dict(grid\_zip)  
 return grid\_dict  
  
  
def grid():  
 *"""() -> dict  
 Create an easy to get grid function.  
 """* return grid\_create(grid\_name(), grid\_measurements(), grid\_id(grid\_measurements()))  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 grid = grid()  
 print(grid)

This will create a matrix with the distance to every other item on the list. This way I can create and refence the matrix and find out if the traveling salesman algorithm is working. The rest of the debugging was mainly with the GUI which was mainly writing code, running it, and seeing what the problems were. With more time, I would like to improve the traveling salesman algorithm and change to a different solution. I would also like to improve the example store or add a real store as the example.

**Code**

from tkinter import \*  
  
  
class StoreRouter:  
  
 def \_\_init\_\_(self):  
 pass  
  
 def main\_frame(self, master):  
 # Set up title of the window  
 master.title('Store Router')  
 # Add a photo to the top of a shopping cart  
 photo = PhotoImage(file="ShoppingCart.png")  
 # Add a frame to be the top frame on the window  
 frame\_top = Frame(master)  
 frame\_top.pack()  
 photo\_label = Label(frame\_top, image=photo)  
 photo\_label.photo = photo  
 photo\_label.pack()  
 # Add three lines of text as an introduction and instructions  
 Label(frame\_top, text='Welcome to Store Router, the app for your routing needs.').pack()  
 Label(frame\_top, text='This is an example store to show how Store Router works and what it does.').pack()  
 Label(frame\_top, text='Check the boxes of the items you would like to buy then hit Route.').pack()  
 # Create frames to make the lists more organized  
 frame = Frame(master)  
 frame.pack()  
 frame2 = Frame(master)  
 frame2.pack()  
 frame3 = Frame(master)  
 frame3.pack()  
 frame4 = Frame(master)  
 frame4.pack()  
 frame5 = Frame(master)  
 frame5.pack()  
 frame6 = Frame(master)  
 frame6.pack()  
 # Create the check boxes and the variable that goes along with them for 1 on and 0 off  
 self.var1 = IntVar()  
 Checkbutton(frame, text="cheese", variable=self.var1).pack(side=LEFT)  
 self.var2 = IntVar()  
 Checkbutton(frame, text="yogurt", variable=self.var2).pack(side=LEFT)  
 self.var3 = IntVar()  
 Checkbutton(frame, text="milk", variable=self.var3).pack(side=LEFT)  
 self.var4 = IntVar()  
 Checkbutton(frame, text="chicken", variable=self.var4).pack(side=LEFT)  
 self.var5 = IntVar()  
 Checkbutton(frame, text="beef", variable=self.var5).pack(side=LEFT)  
 self.var6 = IntVar()  
 Checkbutton(frame2, text="pork", variable=self.var6).pack(side=LEFT)  
 self.var7 = IntVar()  
 Checkbutton(frame2, text="tv dinners", variable=self.var7).pack(side=LEFT)  
 self.var8 = IntVar()  
 Checkbutton(frame2, text="ice cream", variable=self.var8).pack(side=LEFT)  
 self.var9 = IntVar()  
 Checkbutton(frame2, text="waffles", variable=self.var9).pack(side=LEFT)  
 self.var10 = IntVar()  
 Checkbutton(frame2, text="cereal", variable=self.var10).pack(side=LEFT)  
 self.var11 = IntVar()  
 Checkbutton(frame3, text="coffee", variable=self.var11).pack(side=LEFT)  
 self.var12 = IntVar()  
 Checkbutton(frame3, text="tea", variable=self.var12).pack(side=LEFT)  
 self.var13 = IntVar()  
 Checkbutton(frame3, text="bread", variable=self.var13).pack(side=LEFT)  
 self.var14 = IntVar()  
 Checkbutton(frame3, text="cake", variable=self.var14).pack(side=LEFT)  
 self.var15 = IntVar()  
 Checkbutton(frame3, text="crackers", variable=self.var15).pack(side=LEFT)  
 self.var16 = IntVar()  
 Checkbutton(frame4, text="cookies", variable=self.var16).pack(side=LEFT)  
 self.var17 = IntVar()  
 Checkbutton(frame4, text="water", variable=self.var17).pack(side=LEFT)  
 self.var18 = IntVar()  
 Checkbutton(frame4, text="soda", variable=self.var18).pack(side=LEFT)  
 self.var19 = IntVar()  
 Checkbutton(frame4, text="juice", variable=self.var19).pack(side=LEFT)  
 self.var20 = IntVar()  
 Checkbutton(frame4, text="vegetables", variable=self.var20).pack(side=LEFT)  
 self.var21 = IntVar()  
 Checkbutton(frame5, text="fruit", variable=self.var21).pack(side=LEFT)  
 self.var22 = IntVar()  
 Checkbutton(frame5, text="salads", variable=self.var22).pack(side=LEFT)  
 self.var23 = IntVar()  
 Checkbutton(frame5, text="nuts", variable=self.var23).pack(side=LEFT)  
 self.var24 = IntVar()  
 Checkbutton(frame5, text="jam", variable=self.var24).pack(side=LEFT)  
 self.var25 = IntVar()  
 Checkbutton(frame5, text="peanut butter", variable=self.var25).pack(side=LEFT)  
 self.var26 = IntVar()  
 Checkbutton(frame6, text="paper towels", variable=self.var26).pack(side=LEFT)  
 self.var27 = IntVar()  
 Checkbutton(frame6, text="tooth paste", variable=self.var27).pack(side=LEFT)  
 self.var28 = IntVar()  
 Checkbutton(frame6, text="laundry detergent", variable=self.var28)  
 # Create our quit button and our button that creates the results  
 Button(master, text="Route", command=self.result\_frame).pack()  
 Button(master, text="Quit", command=quit).pack()  
  
 def result\_frame(self):  
 # Create a new window with displaying the results  
 result = Toplevel()  
 # Create the window title  
 result.title('Shopping list')  
 # Create a list that takes the variables from the last window  
 v\_list = []  
 # Was having problems with the for statement so set up an x to go through the list  
 x = 0  
 while x < 28:  
 query = [self.var1, self.var2, self.var3, self.var4, self.var5, self.var6, self.var7, self.var8, self.var9,  
 self.var10, self.var11, self.var12, self.var13, self.var14, self.var15, self.var16, self.var17,  
 self.var18, self.var19, self.var20, self.var21, self.var22, self.var23, self.var24, self.var25,  
 self.var26, self.var27, self.var28]  
 # Get all of the 1 or 0's from last window  
 v\_list.append(query[x].get())  
 x += 1  
 # Create a list based off of our v\_list  
 grocery\_list = []  
 # The grid of our grocery store  
 grid = {'cheese': (0, 1), 'milk': (0, 2), 'yogurt': (0, 3), 'chicken': (0, 4), 'beef': (0, 5), 'pork': (0, 6),  
 'tv dinners': (0, 7), 'ice cream': (0, 8), 'waffles': (0, 9), 'cereal': (0, 1), 'coffee': (3, 1),  
 'tea': (3, 2), 'bread': (3, 3), 'cake': (3, 4), 'crackers': (3, 5), 'cookies': (3, 6), 'water': (3, 7),  
 'soda': (3, 8), 'juice': (3, 9), 'vegetables': (6, 1), 'fruit': (6, 1), 'salads': (6, 2),  
 'nuts': (6, 3), 'jam': (6, 4), 'peanut butter': (6, 5), 'paper towels': (6, 6) ,'tooth paste': (6, 7),  
 'laundry detergent': (6, 8)}  
 # List of only the food in order  
 food\_lst = ['cheese', 'milk', 'yogurt', 'chicken', 'beef', 'pork', 'tv dinners', 'ice cream', 'waffles',  
 'cereal', 'coffee', 'tea', 'bread', 'cake', 'crackers', 'cookies', 'water', 'soda', 'juice',  
 'vegetables', 'fruit', 'salads', 'nuts', 'jam', 'peanut butter', 'paper towels', 'tooth paste',  
 'laundry detergent']  
 index = 0  
 # Create a list to get the coordinates from our grocery list in order  
 shopping\_coord = []  
 # Get a list of the food we are looking for as strings  
 g\_food\_list = []  
 # Create a list we will use in our tsp  
 shopping\_matrix = []  
 while index < 28:  
 if v\_list[index] == 1:  
 # Populate our three lists  
 grocery\_list.append(grid[food\_lst[index]])  
 shopping\_matrix.append(grid[food\_lst[index]])  
 g\_food\_list.append(food\_lst[index])  
 index += 1  
 # Start at the entrance or (0, 0)  
 start\_coord = (0, 0)  
 # Create a large number to go back to each run of the tsp  
 smallest\_distance = 100000  
 # Our first run of our greedy tsp for just the entrance  
 for item in shopping\_matrix:  
 if item == start\_coord:  
 continue  
 else:  
 # Our Manhattan distance alg  
 index = 0  
 # Starting coordinate  
 coord1 = (0, 0)  
 # Break it into x and y  
 coord\_x1 = coord1[0]  
 coord\_y1 = coord1[1]  
 # Ending coordinate  
 coord2 = shopping\_matrix[index]  
 # Break it into x and y  
 coord\_x2 = coord2[0]  
 coord\_y2 = coord2[1]  
 # See if x of end is equal to x of start  
 if coord\_x2 == coord\_x1:  
 # If it is distance is a straight line use normal distance formula  
 distance = ((coord\_x1 - coord\_x2) \*\* 2 + (coord\_y1 - coord\_y2) \*\* 2) \*\* 0.5  
 else:  
 # If start of y is greater than 4 then you should go to the top of the isle  
 if coord\_y1 > 4:  
 # Manhattan distance  
 distance = (10 - coord\_y1) + abs(coord\_x2 - coord\_x1) + (10 - coord\_y2)  
 # If start of y is less than or equal to 4 the go to the bottom of the isle  
 elif coord\_y1 <= 4:  
 # Manhattan distance  
 distance = coord\_y1 + abs(coord\_x2 - coord\_x1) + coord\_y2  
 if distance < smallest\_distance:  
 # Change smallest distance to the smallest then make that item the next item  
 smallest\_distance = distance  
 next\_item = shopping\_matrix[index]  
 index += 1  
 # Go through the list and run it through greedy tsp  
 while len(shopping\_matrix) > 0:  
 # Add what we went to into our final list  
 shopping\_coord.append(next\_item)  
 # Remove it from our list so that it no longer runs in the tsp  
 shopping\_matrix.remove(next\_item)  
 # reset smallest distance  
 smallest\_distance = 100000  
 index = 0  
 for coord in shopping\_matrix:  
 if coord == next\_item:  
 continue  
 else:  
 # Distance alg  
 coord1 = shopping\_matrix[0]  
 coord\_x1 = coord1[0]  
 coord\_y1 = coord1[1]  
 coord2 = shopping\_matrix[index]  
 coord\_x2 = coord2[0]  
 coord\_y2 = coord2[1]  
 if coord\_x2 == coord\_x1:  
 distance = ((coord\_x1 - coord\_x2) \*\* 2 + (coord\_y1 - coord\_y2) \*\* 2) \*\* 0.5  
 else:  
 if coord\_y1 > 4:  
 distance = (10 - coord\_y1) + abs(coord\_x2 - coord\_x1) + (10 - coord\_y2)  
 elif coord\_y1 <= 4:  
 distance = coord\_y1 + abs(coord\_x2 - coord\_x1) + coord\_y2  
 if distance < smallest\_distance:  
 # Change smallest distance to the smallest then make that item the next item  
 smallest\_distance = distance  
 next\_item = shopping\_matrix[index]  
 # Make an ending list of groceries as Strings  
 shopping\_list = []  
 index = 0  
 for numbers in shopping\_coord:  
 if grocery\_list[index] == shopping\_coord[index]:  
 # Populate our list of groceries  
 shopping\_list.append(g\_food\_list[index])  
 index += 1  
 # Create our string to add as a label  
 shopping\_list\_string = '\n'.join(shopping\_list)  
 # Start out string of coordinates, start it with a new line ot format with the shopping\_list\_string  
 shopping\_coord\_list = '\n'  
 for coord\_pair in shopping\_coord:  
 # Populate the string  
 shopping\_coord\_string = str(coord\_pair) + '\n'  
 shopping\_coord\_list += shopping\_coord\_string  
 # Create Frames for formatting  
 frame\_top = Frame(result)  
 frame\_top.pack()  
 frame\_middle = Frame(result)  
 frame\_middle.pack()  
 frame\_bottom = Frame(result)  
 frame\_bottom.pack()  
 # Create our text explaining the list  
 Label(frame\_top, text='Here is your list with the top being the first item and the bottom being your '  
 'last.').pack()  
 Label(frame\_top, text='Each item has a coordinate the first number being isle the second being the '  
 'section.').pack()  
 # Create our list  
 Label(frame\_middle, text=shopping\_list\_string).pack(side=LEFT)  
 Label(frame\_middle, text=shopping\_coord\_list).pack(side=RIGHT)  
 # Create an exit  
 Button(frame\_bottom, text="OK", command=quit).pack(side=BOTTOM)  
  
  
# Run the program  
master = Tk()  
app = StoreRouter()  
app.main\_frame(master)  
master.mainloop()