Project Overview

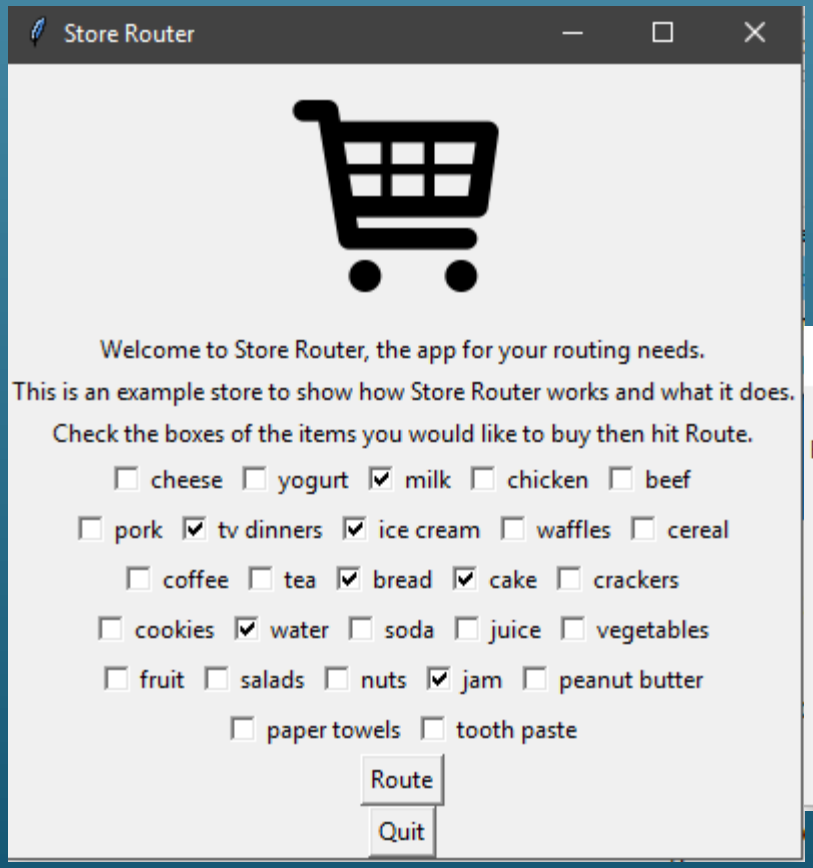
Store Router

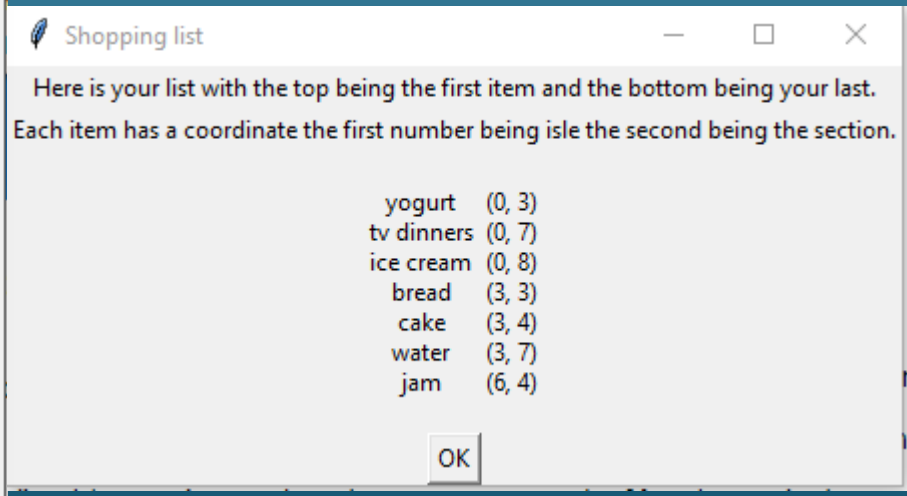
Justin D. Minsk

Store Router

- Create an application that finds the smallest distance to travel in a store using a list created by the user.
- Other applications include 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 your shopping.

Store Router





What the code does

Here we set up the application with our class and our __init__ definition. Then we set up our first frame of window.

```
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()
```

What the code does

Here is and example of some of the formatting for the frame.

```
# Create frames to make the lists more organized
frame = Frame(master)
frame.pack()
frame2 = Frame(master)
frame2.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)
```

```
# 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()
```

What the code does

Here is how we open the next frame and start creating lists.

```
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
```

What the code does

```
# 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
```

What the code does

Here is the first greedy traveling salesman problem. This one is special since it starts at the entrance. This is also the start of our distance problem.

```
# 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]
```

What the code does

Here is the rest of the distance algorithm and the rest of our greedy traveling salesman problem algorithm.

```
# 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
    if 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
```

What the codes does

```
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)
                if 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
```

What the code does

Here we set up our last list and start formatting for the result screen.

```
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
```

What the code does

```
# 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)
```

Demonstration

Questions