CS 2316

Individual Homework 6 – Box Packer

Due: Wednesday, October 9th, before 11:55 PM
Out of 100 points

Files to submit: **1. HW6.py**

For Help:

- TA Helpdesk – Schedule posted on class website.
- Email TA's or use T-Square Forums

Notes:

- **Don't forget to include the required comments and collaboration statement (as outlined on the course syllabus).**
- ***Do not wait until the last minute*** *to do this assignment in case you run into* problems.
- **Read the entire specifications document before starting this assignment.**

Premise

This homework will have you packing rectangular objects into a 2D space, similar to packing boxes into a flat truck. You will be given the dimensions of the space you are packing objects into, as well as a list of boxes. With this information, you will implement and analyze two algorithms that will pack the boxes into the truck: the first which packs the largest boxes into the truck first, and the second which packs the smallest boxes into the truck first. Your program will then report to the user how many boxes were packed, which boxes were packed, which boxes were not packed, and how much space on the truck is left unfilled, using a GUI that you will create.

Homework 6 is a problem-solving homework; we want you to reason through how to develop the algorithms and place things in the GUI, so **do not wait until the last minute to start this assignment**, as this assignment requires you to implement a large amount of functionality.

This homework has an opportunity for **substantial extra credit**, which is described at the end of this document.

File Format Information

You will be required to read in a CSV file of data about the boxes you will be packing.

The format for each row is as follows:

BOX ID, LENGTH, WIDTH

BOX ID is a string that uniquely identifies each box that you will try to place on the truck. It is a string of letters and numbers that may be of any length, such as X5QR76.

LENGTH is the length of the box, in feet. This number should be an integer, such as 15. The length of the box is analogous to the number of rows in your implementation.

WIDTH is the width of the box, in feet. This number should be an integer, such as 15. The width is analogous to the number of columns in your implementation.

A sample row in the file would be

X5QR76, 15, 15

There is no limit to the number of rows that may be contained within the file.

"The Truck"

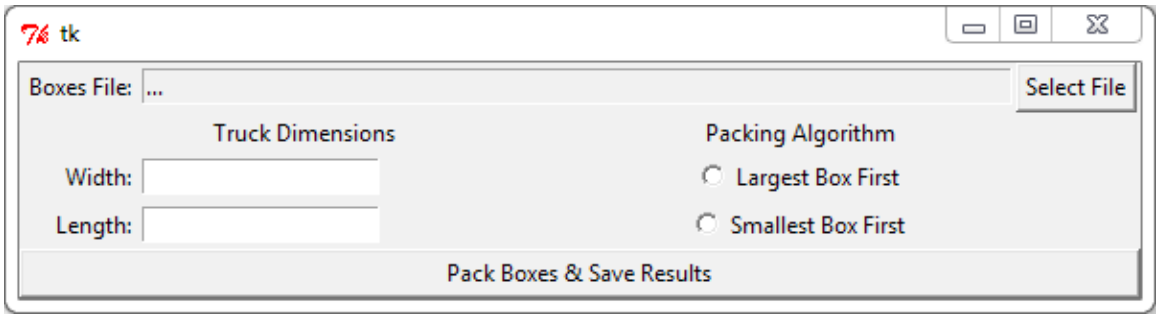
For this assignment, we're going to be packing boxes only one layer high (i.e. in two dimensions only), so we will represent the truck as a 2-dimensional list. ***This 2-dimensional list should be stored as an instance variable named "truck" so that all methods you write will be able to access the 2D list using "self.truck".***

The 2D list will have a cell for each square foot present in your truck; an empty cell will be represented by an empty string, while an occupied cell will be represented by the Box ID of the box that occupies that cell. Represented as a table, a possible truck could look like the following:

"X5QR76"	"X5QR76"	"123XYZ"	"123XYZ"	""	""	""	""
"X5QR76"	"X5QR76"	""	""	""	""	""	""
""	""	""	""	""	""	""	""
""	""	""	""	""	""	""	""

Method Name: **__init__**

This method is automatically called whenever you create a new instance of your GUI class. The __init__ method is responsible for arranging and initializing your GUI. You should create a GUI which looks like the following:

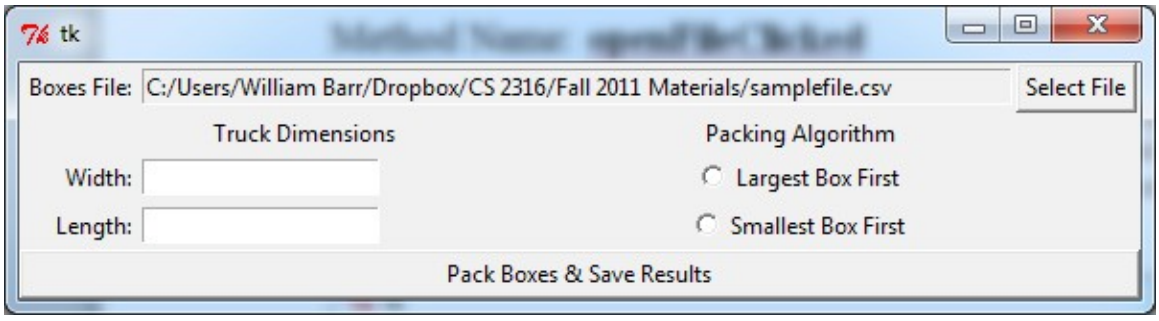


You may generate this window however you wish, but it is *highly* suggested that you use the grid layout manager. Things to note about the GUI you see:

- The entry box between the "Boxes File:" label and the "Select File" button is width 75, readonly, and has text "…" if no file has been selected.
- Clicking on the "Select File" button should call the openFileClicked method.
- Largest Box First and Smallest Box First are Radiobutton widgets. When the window is first opened, neither of these radio buttons should be selected.
- The width and width entry boxes are the default width and have a state of NORMAL.
- The "Pack Boxes & Save Results" button should span the width of the entire window (Hint: To make something like a button or entry box expand to take up the whole space, sticky it in both the east and west directions when using the grid layout manager). Clicking on this button calls the packNSaveClicked method.

Method Name: **openFileClicked**

This method will simply launch a file open dialog box and prompt the user to select a file. If the user selects a file, the text for the Entry box beside the "Boxes file:" label will be set to the file the user chose. If the user chooses cancel, you must set the text of this entry box to be "…". ***Store the file name returned in the "fileName" object variable, so that other methods (such as readBoxsFile) can access the string using "self.fileName".***



Method Name: **readBoxesFile**

Parameters:

None
Return Value:
None

Description:

This method will open the file which the user specified in the open file dialog box, read in the CSV file, and store the data that is read from the file as an object variable which points to a list of lists. ***Name this object variable "boxes" so that other methods can access it using "self.boxes".*** As you read in each row from the CSV file, you should convert the width and length data into integers before saving the row.

In the event that a piece of data in the CSV file is malformed (doesn't have all the required data or has a non-integer width and/or length) or the user did not give you a valid CSV file, you should raise a ValueError which you will catch in the packAndSaveResults method.

Remember to close the file when you are finished.

Method Name: **isValidLocation**

Parameters:

Integer – The row number of the cell you're testing
Integer – The column number of the cell you're testing
Integer – The width of the box you're trying to pack
Integer – The length of the box you're trying to pack

Return Value:

Boolean – True if the box can be packed into the truck starting at location row, col, False if it cannot be packed starting at that location.

Description:

This method will check to see if you can pack a box (with certain dimensions that you passed in) into the truck, with the box's upper left corner starting at the row, column position that was passed in. Do not change the orientation of the box when trying to check and see if the location is valid. (Your union does not allow you to rotate boxes!) The method should return True if none of the cells in the truck that would be occupied by the box are used by another box, and if the box can fit into the truck at that location (i.e. doesn't extend beyond the boundaries of the truck). This method should return False if there is any overlap or if the box would extend beyond the boundaries of the truck.

Method Name: **fillTruckLocation()**

Parameters:

Integer – The row of the space where you wish to start packing the box.
Integer – The column of the space where you wish to start packing the box.
List – The list of box information

Return Value:
None

Description:
This method will accept a row and column, which represents the location where the upper left corner of the box will be located. You must fill in spaces starting at row, col and expanding to the right the width of the box and expanding down the length of the box. You should fill in each cell with the Box ID of the box you are filling the space with.

Method Name: **packBox()**

Parameters:
List – A list containing the data for one particular box.

Return Value:
A Boolean – True if the box was successfully packed into the truck, False if it was not.

Description:
This method will look through the truck and try to find the first empty cell, starting with the first row and first column, then moving to the right on that row until there are no more cells to check, then moving to the second row, first column, then moving to the right until there are no more cells to check on that row, and so on and so forth.

When you find an empty cell, you will attempt to pack the upper left-corner of the box into this position, and the rest of the box will extend to the right and down; use the isValidLocation method you wrote to see if the box will fit in this location. If it will fit, call the fillTruckLocation method with the appropriate information and return True. If the box will not fit into this location, continue searching for an empty cell until you find another one. Repeat this process of finding empty cells and checking to see if the box will fit at that location until you either find a location where the box will fit or you run out of empty locations in the truck. If you have checked all of the empty locations in the truck and none are valid, or if there are no empty locations left, you should return False.

Method Name: **packTruck()**

Parameters:
None

Return Value:
A List – A list of all boxes from the CSV file that were not packed into the truck with the given algorithm.

Description:
This method will attempt to pack all the boxes from the CSV file into the truck.

The order in which it will attempt to place boxes into the truck is dependent upon which radio button the user selected:

If the user picked “Largest Box First”, you will attempt to pack the boxes such that the box with the largest area is packed on the truck first, followed by the second largest area, etc. If two boxes have the same area, prefer boxes that have a “larger” Box ID (as determined by the python greater than operator).

If the user picked “Smallest Box First”, you will attempt to pack the boxes such that the box with the smallest area is packed on the truck first, followed by the second smallest area, etc. If two boxes have the same area, prefer boxes that have a “smaller” Box ID (as determined by the python less than operator).

When you have attempted to pack all the boxes, you should return a list of boxes that were not packed into the truck.

Method Name: **writeTruckToCSV()**

Parameters:

None

Return Value:

None

Description:

This method will write out the contents of the truck object variable as a CSV file. You should be able to open this CSV file in a program such as Excel or LibreOffice Calc and see the “table” view of the boxes packed into your truck. You do not need to format the data in any special way before writing it out. You should save this file as “truckview.csv”.

Use of csv.writer will make this method trivial to implement. Remember to close the file when you are finished.

Function Name: **packNSaveClicked**

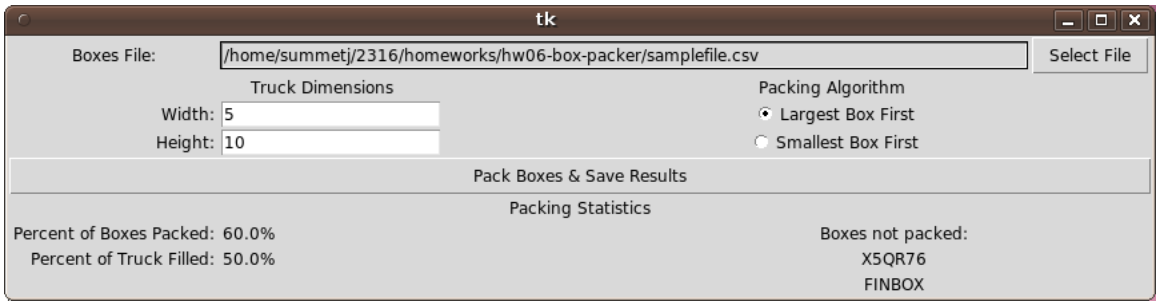
This function will set up the truck object variable to have the appropriate number of empty spaces, read in the file containing information about the boxes, pack the boxes using the algorithm the user specified, write the contents of the truck out to the “truckview.csv” file, and add some widgets to the GUI to show the user some statistics about the truck they just packed.

First, you must make sure that the user has entered an appropriate width and length for the truck as well as selected a packing algorithm. If the user did not provide all of this information (or the user entered an invalid value for width or length), you should pop up a message dialog box informing the user of the mistake.

If readBoxesFile raises a ValueError when you call it, you should display a

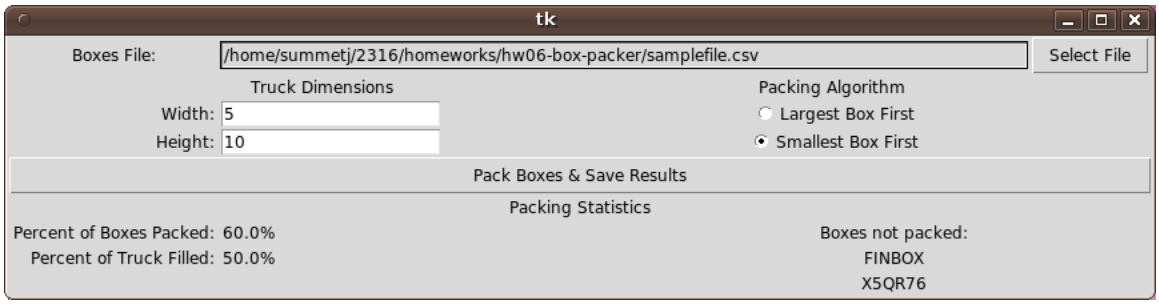
message dialog box to the user informing them that the CSV file is invalid. You should not attempt to pack any boxes if this occurs.

When you have successfully packed as many boxes as possible using the user's specified algorithm, you should update the GUI so that it looks like the following:



You will need to compute the percentage of the boxes packed as well as the percentage of the truck that was filled during the packing process. Additionally, you will need to display the Box IDs of all boxes that were not packed.

Below is another test-case output so you can validate your program:



Extra Credit Opportunity:

If you examine the algorithm and packing procedure we described above, you will notice that this is a rather naïve way to pack the truck. Sometimes, if we had changed the order in which we packed boxes, or didn't necessarily always pack the largest or smallest box, we could find a more optimal solution. Also, we did not permit any kind of transformation of the box, such as a rotation. In some cases, rotating the box would permit the box to be packed.

For extra credit, come up with your own packing algorithm and method to fill in spaces in the truck; there are no restrictions on what you can do to pack the boxes this time around. Your objective is maximize the percentage of boxes packed, the percentage of the truck that is filled, or (optimally) both! The TAs will award extra credit based on the creativity of the approach as well as the results; the

better your algorithm is at maximizing the number of boxes packed or the amount of the truck filled, the more extra credit you will receive. Make sure that the basics of your program work correctly; if your extra credit attempt breaks the base functionality, you will lose points!

If you choose to do this, you need to modify your program so you have three radio buttons instead of two; the third radio button should be labeled appropriately, and when a user selects it, it should pack the boxes using your packing algorithm and method. At the top of your file, write a brief description of how your algorithm and method work and why it is better than the two algorithms we provided.

Be sure to comment your custom algorithm thoroughly!

Grading:

You will earn points as follows for each piece of functionality that works correctly according to the specifications.

The GUI		**20**
GUI has all required initial components	10	
openFileClicked works as described	5	
GUI has proper error checking	5	
readBoxesFile()		**6**
Successfully reads in all information from file	2	
Stores data from file in an instance variable	2	
Converts width and length into integers	2	
isValidLocation()		**10**
Checks truck boundaries	5	
Correctly checks all relevant spaces in truck	5	
fillTruckLocation()		**7**
Correctly fills all spaces in the truck with given dimensions	7	
packBox()		**8**
Correctly searches for empty spaces	5	
Correctly calls fillTruckLocation when space found	3	
packTruck()		**25**
Keeps track of unpacked boxes correctly	10	
Attempts to execute correct packing algorithm	5	
Packs boxes in correct order	10	
writeCSVFile()		**4**
Writes CSV file correctly	4	
packNSaveClicked()		**20**
Adds new GUI elements after computation	10	
Does not reimplement functionality from other methods	3	
Percent packed and percent empty computed correctly	7	