Trolley Packing Problem Write-up

**Implementation**

My implementation of the Trolley Packing Problem consists of 5 java classes. 3 of the classes are used to construct parcel, pile and trolley objects. One of the remaining 2 classes is the ‘Packer’ class, which contains the main algorithm for packing parcels according to the coursework specification. The final class is a test class used for testing and analysing the efficiency of my online and offline algorithms.

The Parcel class defines the height, width and custNo (Customer ID) of created parcels. The Comparable interface is implemented by this class and, coupled with the compareTo method I wrote, allows Parcel objects to be sorted when the offline version of the algorithm is run.

The Pile class defines the width, the width of the last parcel added to the pile (the top parcel), the customerID of the pile and the current height of the pile. Each Pile object contains a linked list to store parcels in, I decided that this implementation made the most sense.

The Trolley class defines the height, width, customerIDs and width left (remaining width) in the trolley. The customerIDs variable is an array of integers sized 2, the maximum number of customer’s parcels allowed in any one trolley. The class also contains a linked list for storing piles. Two notable methods in this class are the setCustArray method and the idValid method. The setCustArray method takes a parcel’s customerID as a parameter and adds it to the custIDs array, where there is space. The idValid method is used for validation by the main packing algorithm when attempting to place a parcel in a trolley, it checks whether there are already parcels belonging to that customer in the trolley or if there are currently less than 2 customer’s with parcels in the trolley.

There are getter and setter methods throughout my classes to allow for interaction between them.

The Packer class contains a number of variables that are used to determine the maximum and minimum heights of parcels and trolleys as well as handling the randomisation of these dimensions. The class contains 2 linked lists, one to store the initially created parcels and another to store created trolleys. The howManyParcels variable is used to set how many parcels should be created and equally how many corresponding trolleys. The parcelAdded boolean variable is used to indicate when a parcel has successfully been placed and allows the algorithm to know when to move onto the next parcel. The trolleysUsed variable is used by the countTrolleysUsed method to count how many trolleys the algorithm has used in packing a set of parcels. The method iterates through all of the created trolleys and determines whether a trolley has been used (by comparing the width of the trolley with the width left in the trolley). If the trolley has been used then these two values will not be equal. The createRandomParcels method creates random values, within the range stated by the minimum and maximum variables in the class, for which to give each parcel. The method then adds the parcels to the initialParcels linked list. The createTrolleys method creates trolleys according to the set dimensions in the Trolley class; the created Trolley objects are added to the trolleys linked list as they are created.

The onlineAlgorithm method performs the actual packing of parcels. A series of list iterators allow for the retrieval of objects from their linked list data structures. To start, the algorithm gets the first parcel in the initialParcels linked list and the first trolley in the trolleys linked list. A while loop tells the algorithm to run for each parcel in the initialParcels list. Whenever a parcel is successfully added, the parcelAdded boolean variable is set to true. This stops the packing for this parcel and allows the algorithm to continue onto placing the next parcel.

If there are no piles already in the retrieved trolley, a pile is created and its values set to reflect the parcel that has started the pile. If there are already existing piles in the trolley, the algorithm checks each pile in the trolley until the validation criteria (parcel fits widthways, the new pile height will not exceed the height of the trolley and the customerID of the parcel matches that of the existing pile) is met. The algorithm iterates through piles until the parcel meets the validation criteria or creates a new pile if the parcel cannot be placed in any of those already existing and if there is room for a new pile to be placed in the trolley. If there is no room to create a new pile in the current trolley then the algorithm iterates to the next trolley in the list and repeates the algorithm, looking for where the parcel can be placed whilst satisfying all of the constraints detailed by the specification. Once a parcel is placed, the algorithm iterates to the next parcel in the initialParcels list and then runs again. This process repeats until all parcels have been packed.

The offlineAlgorithm method simply sorts the linked list of initial parcels, using the comparator I wrote in the Parcels class, and then calls the onlineAlgorithm.

My Packer class also contains a printTrolleys method that can be called to print all of the trolleys, piles and parcels created by the algorithm(s). This method is useful in testing that my implementation actually meets the requirements of the coursework specification. Currently, my Test class contains a call to this method but is commented out for console presentation purposes.

**Testing**

To test the performance of both my online and offline algorithms, I created a Test class. The bulk of the code inside this class is contained within a for loop so that multiple tests can be run on the algorithms for more taking an average and obtaining more accurate results. Firstly, parcels and trolleys are created. The method calls to both the online and offline algorithms are sandwiched between variable assignments, which note the time before the algorithm starts and the time when it finishes running. By deducting the end time from the start time, the amount of time taken by the packing algorithm to complete is determined.

At the beginning of each test iteration, the clearPT method is called; this clears both the set of parcels for packing and any created trolleys. If this operation were not performed then the algorithm would repeatedly place the same set of parcels into non-empty trolleys, making any yielded test data redundant. After running the online algorithm and before the offline algorithm is started, the clearTrolleys method is called. The trolleys need to be cleared so that the offline algorithm has a fresh set of trolleys in which to place parcels, rather than a set filled with parcels previously placed by the online algorithm in the same test iteration.

My Test class updates several variables on each iteration of the for loop (for each test cycle), these variables sum up the total time taken and total number of trolleys used by each algorithm over the course of however many test cycles are performed. When the testing is completed these values are divided by the number of tests performed, which gives the average time taken and number of trolleys used by each algorithm. These results are then printed along with a comparison of efficiency between the online and offline algorithms, calculated from the averages and presented as percentage values.

**Correctness Testing**

To test the correctness of my implementation I ran a test, creating 10 random parcels and packing them using the online and offline algorithms. Using a method named printInitialParcels, I printed out these 10 randomly generated parcels in order to see the dimensions and customerIDs which each had been given. I then used my printTrolleys method to determine where the parcels had been placed. To check that the algorithms had correctly performed the packing procedure, I traced the packing process by drawing out the 10 trolleys and the structure in which the initial parcels should be placed in them.

The 10 random parcels:

Parcel Height: 108 Parcel Width: 326 Customer No: 3

Parcel Height: 850 Parcel Width: 1353 Customer No: 1

Parcel Height: 491 Parcel Width: 1677 Customer No: 2

Parcel Height: 1349 Parcel Width: 1353 Customer No: 1

Parcel Height: 1602 Parcel Width: 1812 Customer No: 2

Parcel Height: 175 Parcel Width: 1937 Customer No: 2

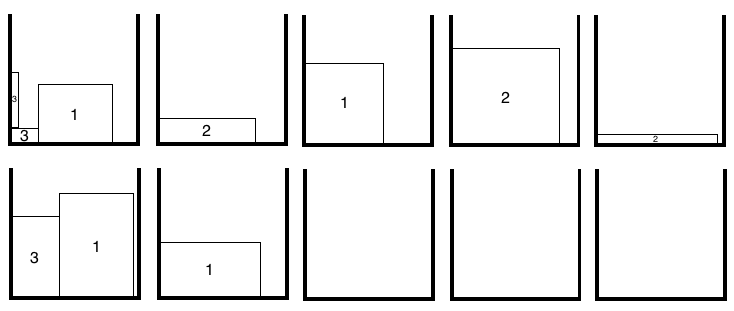
Parcel Height: 1404 Parcel Width: 748 Customer No: 3

Parcel Height: 1671 Parcel Width: 1181 Customer No: 1

Parcel Height: 1088 Parcel Width: 187 Customer No: 3

Parcel Height: 766 Parcel Width: 1779 Customer No: 1

Online Algorithm Trace



Trolleys after running the online algorithm:

Trolley Number: 1

Trolley Height: 2000 Trolley Width: 2000 Width Left: 321 Piles in trolley: 2

Pile width: 326 Last width: 187 Customer ID: 3 Current height: 1196 Parcels in pile: 2

Parcel Height: 108 Parcel Width: 326 Customer No: 3

Parcel Height: 1088 Parcel Width: 187 Customer No: 3

Pile width: 1353 Last width: 1353 Customer ID: 1 Current height: 850 Parcels in pile: 1

Parcel Height: 850 Parcel Width: 1353 Customer No: 1

--------------------------------------------------------------

Trolley Number: 2

Trolley Height: 2000 Trolley Width: 2000 Width Left: 323 Piles in trolley: 1

Pile width: 1677 Last width: 1677 Customer ID: 2 Current height: 491 Parcels in pile: 1

Parcel Height: 491 Parcel Width: 1677 Customer No: 2

--------------------------------------------------------------

Trolley Number: 3

Trolley Height: 2000 Trolley Width: 2000 Width Left: 647 Piles in trolley: 1

Pile width: 1353 Last width: 1353 Customer ID: 1 Current height: 1349 Parcels in pile: 1

Parcel Height: 1349 Parcel Width: 1353 Customer No: 1

--------------------------------------------------------------

Trolley Number: 4

Trolley Height: 2000 Trolley Width: 2000 Width Left: 188 Piles in trolley: 1

Pile width: 1812 Last width: 1812 Customer ID: 2 Current height: 1602 Parcels in pile: 1

Parcel Height: 1602 Parcel Width: 1812 Customer No: 2

--------------------------------------------------------------

Trolley Number: 5

Trolley Height: 2000 Trolley Width: 2000 Width Left: 63 Piles in trolley: 1

Pile width: 1937 Last width: 1937 Customer ID: 2 Current height: 175 Parcels in pile: 1

Parcel Height: 175 Parcel Width: 1937 Customer No: 2

--------------------------------------------------------------

Trolley Number: 6

Trolley Height: 2000 Trolley Width: 2000 Width Left: 71 Piles in trolley: 2

Pile width: 748 Last width: 748 Customer ID: 3 Current height: 1404 Parcels in pile: 1

Parcel Height: 1404 Parcel Width: 748 Customer No: 3

Pile width: 1181 Last width: 1181 Customer ID: 1 Current height: 1671 Parcels in pile: 1

Parcel Height: 1671 Parcel Width: 1181 Customer No: 1

--------------------------------------------------------------

Trolley Number: 7

Trolley Height: 2000 Trolley Width: 2000 Width Left: 221 Piles in trolley: 1

Pile width: 1779 Last width: 1779 Customer ID: 1 Current height: 766 Parcels in pile: 1

Parcel Height: 766 Parcel Width: 1779 Customer No: 1

--------------------------------------------------------------

Trolley Number: 8

Trolley Height: 2000 Trolley Width: 2000 Width Left: 2000 Piles in trolley: 0

--------------------------------------------------------------

Trolley Number: 9

Trolley Height: 2000 Trolley Width: 2000 Width Left: 2000 Piles in trolley: 0

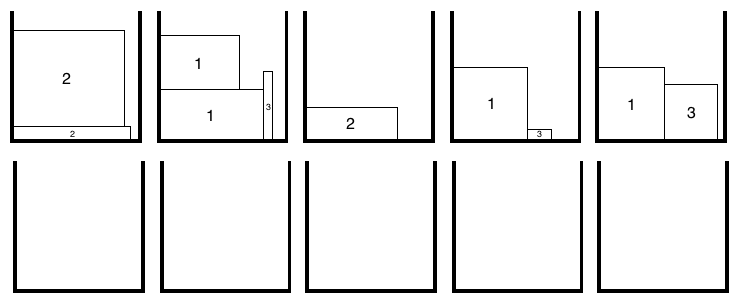
--------------------------------------------------------------

Trolley Number: 10

Trolley Height: 2000 Trolley Width: 2000 Width Left: 2000 Piles in trolley: 0

--------------------------------------------------------------

Offline Algorithm Trace



Trolleys after running the offline algorithm:

Trolley Number: 1

Trolley Height: 2000 Trolley Width: 2000 Width Left: 63 Piles in trolley: 1

Pile width: 1937 Last width: 1812 Customer ID: 2 Current height: 1777 Parcels in pile: 2

Parcel Height: 175 Parcel Width: 1937 Customer No: 2

Parcel Height: 1602 Parcel Width: 1812 Customer No: 2

--------------------------------------------------------------

Trolley Number: 2

Trolley Height: 2000 Trolley Width: 2000 Width Left: 34 Piles in trolley: 2

Pile width: 1779 Last width: 1353 Customer ID: 1 Current height: 1616 Parcels in pile: 2

Parcel Height: 766 Parcel Width: 1779 Customer No: 1

Parcel Height: 850 Parcel Width: 1353 Customer No: 1

Pile width: 187 Last width: 187 Customer ID: 3 Current height: 1088 Parcels in pile: 1

Parcel Height: 1088 Parcel Width: 187 Customer No: 3

--------------------------------------------------------------

Trolley Number: 3

Trolley Height: 2000 Trolley Width: 2000 Width Left: 323 Piles in trolley: 1

Pile width: 1677 Last width: 1677 Customer ID: 2 Current height: 491 Parcels in pile: 1

Parcel Height: 491 Parcel Width: 1677 Customer No: 2

--------------------------------------------------------------

Trolley Number: 4

Trolley Height: 2000 Trolley Width: 2000 Width Left: 321 Piles in trolley: 2

Pile width: 1353 Last width: 1353 Customer ID: 1 Current height: 1349 Parcels in pile: 1

Parcel Height: 1349 Parcel Width: 1353 Customer No: 1

Pile width: 326 Last width: 326 Customer ID: 3 Current height: 108 Parcels in pile: 1

Parcel Height: 108 Parcel Width: 326 Customer No: 3

--------------------------------------------------------------

Trolley Number: 5

Trolley Height: 2000 Trolley Width: 2000 Width Left: 71 Piles in trolley: 2

Pile width: 1181 Last width: 1181 Customer ID: 1 Current height: 1671 Parcels in pile: 1

Parcel Height: 1671 Parcel Width: 1181 Customer No: 1

Pile width: 748 Last width: 748 Customer ID: 3 Current height: 1404 Parcels in pile: 1

Parcel Height: 1404 Parcel Width: 748 Customer No: 3

--------------------------------------------------------------

Trolley Number: 6

Trolley Height: 2000 Trolley Width: 2000 Width Left: 2000 Piles in trolley: 0

--------------------------------------------------------------

Trolley Number: 7

Trolley Height: 2000 Trolley Width: 2000 Width Left: 2000 Piles in trolley: 0

--------------------------------------------------------------

Trolley Number: 8

Trolley Height: 2000 Trolley Width: 2000 Width Left: 2000 Piles in trolley: 0

--------------------------------------------------------------

Trolley Number: 9

Trolley Height: 2000 Trolley Width: 2000 Width Left: 2000 Piles in trolley: 0

--------------------------------------------------------------

Trolley Number: 10

Trolley Height: 2000 Trolley Width: 2000 Width Left: 2000 Piles in trolley: 0

--------------------------------------------------------------

The fact that my traces match the data yielded by my printTrolleys method proves that the algorithms do function correctly.

**Measurements Observed**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Test Number** | **Parcels Created** | **Number of Tests Ran** | **Online Packing Average Time Taken** | **Online Packing Average Number of Trolleys Used** | **Offline Packing Average Time Taken** | **Offline Packing Average Number of Trolleys Used** | **Comparison of Online and Offline Average Time Taken** | **Comparison of Online and Offline Average Number of Trolleys Used** |
| **1** | 1000 | 100 | 3 milliseconds | 316 | 2 milliseconds | 285 | The offline algorithm, on average, took 50% less time than the online algorithm. | The offline algorithm, on average, used 10% less trolleys than the online algorithm. |
| **2** | 10,000 | 100 | 363 milliseconds | 2881 | 235 milliseconds | 2721 | The offline algorithm, on average, took 54% less time than the online algorithm. | The offline algorithm, on average, used 5% less trolleys than the online algorithm. |
| **3** | 10,000 | 1000 | 321 milliseconds | 2883 | 212 milliseconds | 2726 | The offline algorithm, on average, took 51% less time than the online algorithm. | The offline algorithm, on average, used 5% less trolleys than the online algorithm. |
| **4** | 100,000 | 1 | 152943 milliseconds | 27214 | 76172 milliseconds | 26427 | The offline algorithm, on average, took 100% less time than the online algorithm. | The offline algorithm, on average, used 2% less trolleys than the online algorithm. |

**Conclusion of Test Results**

From the tests performed, detailed in the table above, some solid conclusions can be made about the implemented trolley packing algorithms.

The first observation that can be made from these test results is that it seems the more parcels are placed, the larger the difference in time efficiency between the two algorithms. A problem here is that due to processing power constraints, I decided to perform only one test with this volume of parcels. Test number 4, which placed 100,000 parcels, saw the offline algorithm take 100% less time than the online algorithm when placing the same parcel set. The same test resulted in the offline algorithm using just 2% less trolleys than the online algorithm. An identifiable trend across the test results seems to be that the smaller the number of parcels being placed, the more efficient the offline algorithm tends to perform in relation to the number of trolleys used. As the size of the parcel set is increased, the smaller the improvement in the average number of trolleys used (by the offline algorithm) becomes.

In each of the 4 tests, it is clearly evident that the offline packing algorithm is significantly more efficient in placing parcels than non-sorting, online packing algorithm. By ordering the parcels by width alone, largest first, the process of packing the parcels into trolleys witnesses both an increase in speed and a reduction in the number of trolleys required in order to contain parcels while certain constraints exist over the process.

**Graphical Representation of Test Results**

**Pseudo-code**

**Algorithm** onlineAlgorithm()

**Inputs** P: List of Parcels, Q: List of Piles, T: List of Trolleys, custArray: Array of Integers

**Variables** parcelAdded: Boolean, currentParcel: Parcel, currentPile: Pile, currentTrolley: Trolley, newPile: Pile

**Begin**

**while** P has unplaced parcels **do**

**set** parcelAdded to false

**get** next parcel from P

**set** currentParcel to got parcel

**while** parcelAdded = false

**get** nexttrolley from T

**set** currentTrolley to got trolley

**if** there are no piles in currentTrolley

**construct** Pile(currentTrolley) taking parameters from currentTrolley

**set** newPile to created pile

**add** currentParcel to newPile

**add** newPile to currentTrolley

**update** currentTrolley

**set** custArray(currentParcel.customerID)

**set** parcelAdded = true

**else if** there are piles in currentTrolley

**get** next pile from Q

**set** currentPile to got pile

**if** currentParcel fits in currentPile and meets validation

**add** currentParcel to currentPile

**set** parcelAdded = true

**else**

**while** currentParcel does not fit currentPile **do**

**get** next pile from Q

**if** currentParcel fits in currentPile and meets

validation

**add** currentParcel to currentPile

**update** currentTrolley

**set** parcelAdded = true

**od**

**fi**

**if** all piles in Q have been checked

**if** there is room in currentTrolley for a new pile

**construct** Pile(currentTrolley) taking parameters from currentTrolley

**set** newPile to created pile

**add** currentParcel to newPile

**add** newPile to currentTrolley

**update** currentTrolley

**set** custArray(currentParcel.customerID)

**set** parcelAdded = true

**else** iterate to the next trolley in T

**fi**

**fi**

**od**

**od**

**End**

**Algorithm** offlineAlgorithm()

**Inputs** P: List of Parcels

**Begin**

SORT P by width descending

CALL onlineAlgorithm()

**End**