

C950 Task - 2 WGUPS Write-Up

(Task-2: The implementation phase of the WGUPS Routing Program).
(Zip your source code and upload it with this file)

Aaron Tandem

ID #010169998

WGU Email: atandem@wgu.edu

1/20/24

C950 Data Structures and Algorithms II

A. Hash Table

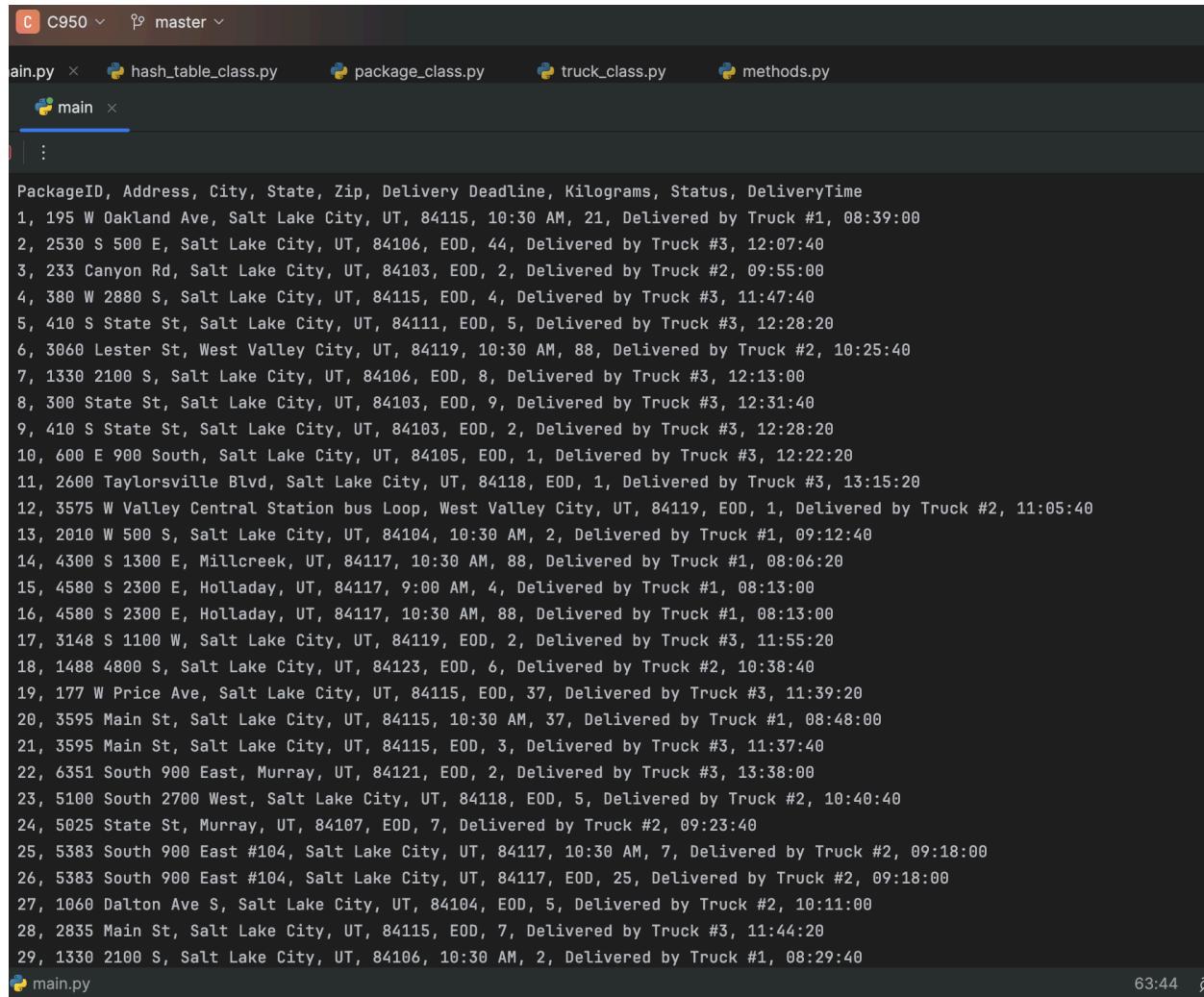
Below is a screenshot of the hash table's code: (full code can be viewed in the provided zip file)

```
102
103
104 # This is the instance of the self-adjusting data structure (hash table), followed by a method call to load it with package data
105 packageHashTable = hash_table_class.ChainingHashTable()
106 methods.loadPackageData( fileName: "WGUPS_package_file.csv", packageHashTable)
```

The screenshot shows a code editor with five tabs at the top: main.py, hash_table_class.py (which is the active tab), package_class.py, truck_class.py, and methods.py. The code in hash_table_class.py is as follows:

```
1 # C950 - Webinar-1 - Let's Go Hashing
2 # W-1_ChainingHashTable_zyBooks_Key-Value.py
3 # Ref: zyBooks: Figure 7.8.2: Hash table using chaining.
4 # Modified for Key:Value
5
6 # HashTable class using chaining.
7 usage  ↳ Aaron Tandem
8 class ChainingHashTable:
9     # Constructor with optional initial capacity parameter.
10    # Assigns all buckets with an empty list.
11    # Aaron Tandem
12    def __init__(self, initial_capacity=40):
13        # initialize the hash table with empty bucket list entries.
14        self.table = []
15        for i in range(initial_capacity):
16            self.table.append([])
17
18    # Inserts a new item into the hash table.
19    # 2 usages (2 dynamic)  ↳ Aaron Tandem
20    def insert(self, key, item): # does both insert and update
21        # get the bucket list where this item will go.
22        bucket = hash(key) % len(self.table)
23        bucket_list = self.table[bucket]
24
25        # update key if it is already in the bucket
26        for kv in bucket_list:
27            # print (key_value)
28            if kv[0] == key:
29                kv[1] = item
30                return True
31
32    # if not, insert the item to the end of the bucket list.
```

Proof the hash table contains correct data, as seen in Option #1 of the user interface:



The screenshot shows a terminal window titled 'C950' with a branch dropdown set to 'master'. The current file is 'main.py'. The code in 'main.py' is as follows:

```

1  PackageID, Address, City, State, Zip, Delivery Deadline, Kilograms, Status, DeliveryTime
2, 195 W Oakland Ave, Salt Lake City, UT, 84115, 10:30 AM, 21, Delivered by Truck #1, 08:39:00
3, 2530 S 500 E, Salt Lake City, UT, 84106, EOD, 44, Delivered by Truck #3, 12:07:40
4, 233 Canyon Rd, Salt Lake City, UT, 84103, EOD, 2, Delivered by Truck #2, 09:55:00
5, 380 W 2880 S, Salt Lake City, UT, 84115, EOD, 4, Delivered by Truck #3, 11:47:40
6, 410 S State St, Salt Lake City, UT, 84111, EOD, 5, Delivered by Truck #3, 12:28:20
7, 3060 Lester St, West Valley City, UT, 84119, 10:30 AM, 88, Delivered by Truck #2, 10:25:40
8, 1330 2100 S, Salt Lake City, UT, 84106, EOD, 8, Delivered by Truck #3, 12:13:00
9, 300 State St, Salt Lake City, UT, 84103, EOD, 9, Delivered by Truck #3, 12:31:40
10, 410 S State St, Salt Lake City, UT, 84103, EOD, 2, Delivered by Truck #3, 12:28:20
11, 600 E 900 South, Salt Lake City, UT, 84105, EOD, 1, Delivered by Truck #3, 12:22:20
12, 2600 Taylorsville Blvd, Salt Lake City, UT, 84118, EOD, 1, Delivered by Truck #3, 13:15:20
13, 3575 W Valley Central Station bus Loop, West Valley City, UT, 84119, EOD, 1, Delivered by Truck #2, 11:05:40
14, 2010 W 500 S, Salt Lake City, UT, 84104, 10:30 AM, 2, Delivered by Truck #1, 09:12:40
15, 4300 S 1300 E, Millcreek, UT, 84117, 10:30 AM, 88, Delivered by Truck #1, 08:06:20
16, 4580 S 2300 E, Holladay, UT, 84117, 9:00 AM, 4, Delivered by Truck #1, 08:13:00
17, 3148 S 1100 W, Salt Lake City, UT, 84119, EOD, 2, Delivered by Truck #3, 11:55:20
18, 1488 4800 S, Salt Lake City, UT, 84123, EOD, 6, Delivered by Truck #2, 10:38:40
19, 177 W Price Ave, Salt Lake City, UT, 84115, EOD, 37, Delivered by Truck #3, 11:39:20
20, 3595 Main St, Salt Lake City, UT, 84115, 10:30 AM, 37, Delivered by Truck #1, 08:48:00
21, 3595 Main St, Salt Lake City, UT, 84115, EOD, 3, Delivered by Truck #3, 11:37:40
22, 6351 South 900 East, Murray, UT, 84121, EOD, 2, Delivered by Truck #3, 13:38:00
23, 5100 South 2700 West, Salt Lake City, UT, 84118, EOD, 5, Delivered by Truck #2, 10:40:40
24, 5025 State St, Murray, UT, 84107, EOD, 7, Delivered by Truck #2, 09:23:40
25, 5383 South 900 East #104, Salt Lake City, UT, 84117, 10:30 AM, 7, Delivered by Truck #2, 09:18:00
26, 5383 South 900 East #104, Salt Lake City, UT, 84117, EOD, 25, Delivered by Truck #2, 09:18:00
27, 1060 Dalton Ave S, Salt Lake City, UT, 84104, EOD, 5, Delivered by Truck #2, 10:11:00
28, 2835 Main St, Salt Lake City, UT, 84115, EOD, 7, Delivered by Truck #3, 11:44:20
29, 1330 2100 S, Salt Lake City, UT, 84106, 10:30 AM, 2, Delivered by Truck #1, 08:29:40

```

The status bar at the bottom right shows '63:44'.

B. Look-Up Functions

Screenshot of Look-up function, as seen in **methods.py**:

```

15
16
17    # This method takes a hashTable and packageID as arguments and returns the correct package from that hashtable for that packageID
18    def hashTableLookUp(hashTable, packageID):
19        return hashTable.search(packageID)
20
21

```

Proof of functionality:

The screenshot shows a code editor with several tabs at the top: main.py (which is active), hash_table_class.py, package_class.py, truck_class.py, and methods.py. The main.py file contains code for a truck class and a methods class, specifically demonstrating the look-up function. The code includes calls to addPackage and deliverTruckPackages methods, and two print statements for look-up tests.

```
170 truck3.addPackage(packageHashTable.search(8))
171 truck3.addPackage(packageHashTable.search(10))
172 truck3.addPackage(packageHashTable.search(11))
173 truck3.addPackage(packageHashTable.search(35))
174 truck3.addPackage(packageHashTable.search(17))
175 truck3.addPackage(packageHashTable.search(19))

176
177 # Method call of deliverTruckPackages() that sends Truck 3 off at the same time the driver for Truck 2
178 deliverTruckPackages(truck3, truck2.getEndingTime().time())
179

180 print("Look-up Function Test:")
181 print(methods.hashTableLookUp(packageHashTable, packageID: 38))
182 print(methods.hashTableLookUp(packageHashTable, packageID: 7))
183
```

The terminal below shows the execution of the main.py script and its output. The output displays the results of the look-up function test, showing two packages found in the hash table.

```
Run main ×

"/Users/owner/Downloads/Online School/WGU Computer Science BSCS/C950/.venv/bin/python" /Users/owner/Downl
↑ Look-up Function Test:
↓
  PackageID, Address, City, State, Zip, Delivery Deadline, Kilograms, Status, DeliveryTime
  38, 410 S State St, Salt Lake City, UT, 84111, EOD, 9, Delivered by Truck #2, 09:51:40
  ↴
  ↴ PackageID, Address, City, State, Zip, Delivery Deadline, Kilograms, Status, DeliveryTime
  ↴ 7, 1330 2100 S, Salt Lake City, UT, 84106, EOD, 8, Delivered by Truck #3, 12:13:00
```

C. Original Code

Below are a few screenshots showing major blocks of code demonstrating implementation of this project: (full code can be viewed in the provided zip file for greater detail)

C950 Task - 2 WGUPS Write-Up

main.py:

```
main.py x hash_table_class.py package_class.py truck_class.py methods.py

7
8
9  # This method finds and returns how many miles there are between two provided addresses held within a list storing address data.
10 # It is subsequently used in the minDistanceFrom() method below to sequentially find which two addresses are closest to one another.
11 # 4 usages ↳ Aaron Tandem
12 def distanceBetween(address1, address2):
13     indexAddress1 = addressDataList.index(address1)
14     indexAddress2 = addressDataList.index(address2)
15
16     # The 2D list is a series of arrays of i++ size, so 'coordinates' will only work uni-directionally
17     # e.g. distanceDataList[3][22] will return an integer representing distance, but distanceDataList[22][3] will throw an exception
18     # However, because coordinates should be 'reversible', logic should be provided for cases when a coordinate is 'out-of-bounds'
19     # So, this 'flips' the indexes when necessary so that the index reference never exceeds what the Address Data List allows for
20     if indexAddress2 > indexAddress1:
21         tempIndex = indexAddress1
22         flippedIndexAddress1 = indexAddress2
23         flippedIndexAddress2 = tempIndex
24         return distanceDataList[flippedIndexAddress1][flippedIndexAddress2]
25
26     else:
27         return distanceDataList[indexAddress1][indexAddress2]
28
29
30     # This method makes use of the distanceBetween() function above to sequentially find the address of a package in a truck's list that is
31     # closest to the provided 'fromAddress' (i.e. 'where the truck was last located'). This 'fromAddress' generally always starts at the WGUPS
32     # HUB located at "4001 South 700 East, Salt Lake City, UT 84107". However, this address is automatically updated to the address of the last
33     # delivered package in the recursive call of minDistanceFrom() in the deliverTruckPackages() method below. This is a critical part of the
34     # Nearest Neighbor Greedy Algorithm used by this program to solve WGUPS's package delivery efficiency issues.
35     # Nearest Neighbor Greedy Algorithm used by this program to solve WGUPS's package delivery efficiency issues.
36     # 1 usage ↳ Aaron Tandem
37
38 > def minDistanceFrom(fromAddress, truckPackages):...
```

```
main.py x hash_table_class.py package_class.py truck_class.py methods.py

63 > def deliverTruckPackages(truck, startTime):...
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104 # This is the instance of the self-adjusting data structure (hash table), followed by a method call to load it with package data
105 packageHashTable = hash_table_class.ChainingHashTable()
106 methods.loadPackageData( fileName: "WGUPS_package_file.csv", packageHashTable)
107
108 # This is an empty list data structure that'll hold distances, followed by a method call to load the list with distance data
109 distanceDataList = []
110 methods.loadDistanceData( fileName: "WGUPS_distance_table.csv", distanceDataList)
111
112 # This is an empty list data structure that'll hold addresses, followed by a method call to load the list with address data
113 addressDataList = []
114 methods.loadAddressData( fileName: "WGUPS_distance_table.csv", addressDataList)
115
116 # Truck 1 will be the first truck to leave the HUB, prioritizing packages that have early deadlines
117 truck1 = truck_class.Truck(1) # The constructor of the Truck Class is called to create a new truck object
118 # Packages are loaded manually by calling the truck's addPackage() along with the hash table's search functionality
119 truck1.addPackage(packageHashTable.search(1))
120 # packages 13, 14, 15, 16, 19, 20 must all be delivered together
121 truck1.addPackage(packageHashTable.search(13))
122 truck1.addPackage(packageHashTable.search(14))
123 truck1.addPackage(packageHashTable.search(15))
124 truck1.addPackage(packageHashTable.search(16))
125 truck1.addPackage(packageHashTable.search(20))
126 # other packages
127 truck1.addPackage(packageHashTable.search(29))
128 truck1.addPackage(packageHashTable.search(38))
129 truck1.addPackage(packageHashTable.search(31))
130 truck1.addPackage(packageHashTable.search(34))
131 truck1.addPackage(packageHashTable.search(37))
132 truck1.addPackage(packageHashTable.search(40))
```

hash_table_class.py:

```
# C950 - Webinar-1 - Let's Go Hashing
# W-1_ChainingHashTable_zyBooks_Key-Value.py
# Ref: zyBooks: Figure 7.8.2: Hash table using chaining.
# Modified for Key:Value

# HashTable class using chaining.

# usage  ↳ Aaron Tandem
class ChainingHashTable:
    # Constructor with optional initial capacity parameter.
    # Assigns all buckets with an empty list.
    ↳ Aaron Tandem
    def __init__(self, initial_capacity=40):
        # initialize the hash table with empty bucket list entries.
        self.table = []
        for i in range(initial_capacity):
            self.table.append([])

    # Inserts a new item into the hash table.
    # 2 usages(2 dynamic) ↳ Aaron Tandem
    def insert(self, key, item): # does both insert and update
        # get the bucket list where this item will go.
        bucket = hash(key) % len(self.table)
        bucket_list = self.table[bucket]

        # update key if it is already in the bucket
        for kv in bucket_list:
            # print (key_value)
            if kv[0] == key:
                kv[1] = item
                return True

        # if not, insert the item to the end of the bucket list.
```

package_class.py:

The screenshot shows a code editor with multiple tabs at the top: main.py, hash_table_class.py, package_class.py (which is the active tab), truck_class.py, and methods.py. The package_class.py tab has a blue background. The code itself is a Python class definition for a 'Package' object. It includes comments explaining its purpose, authorship, and usage. The code uses f-strings for printing and includes several getter and setter methods. There are several yellow lightbulb icons placed throughout the code, likely indicating potential refactoring or unused imports.

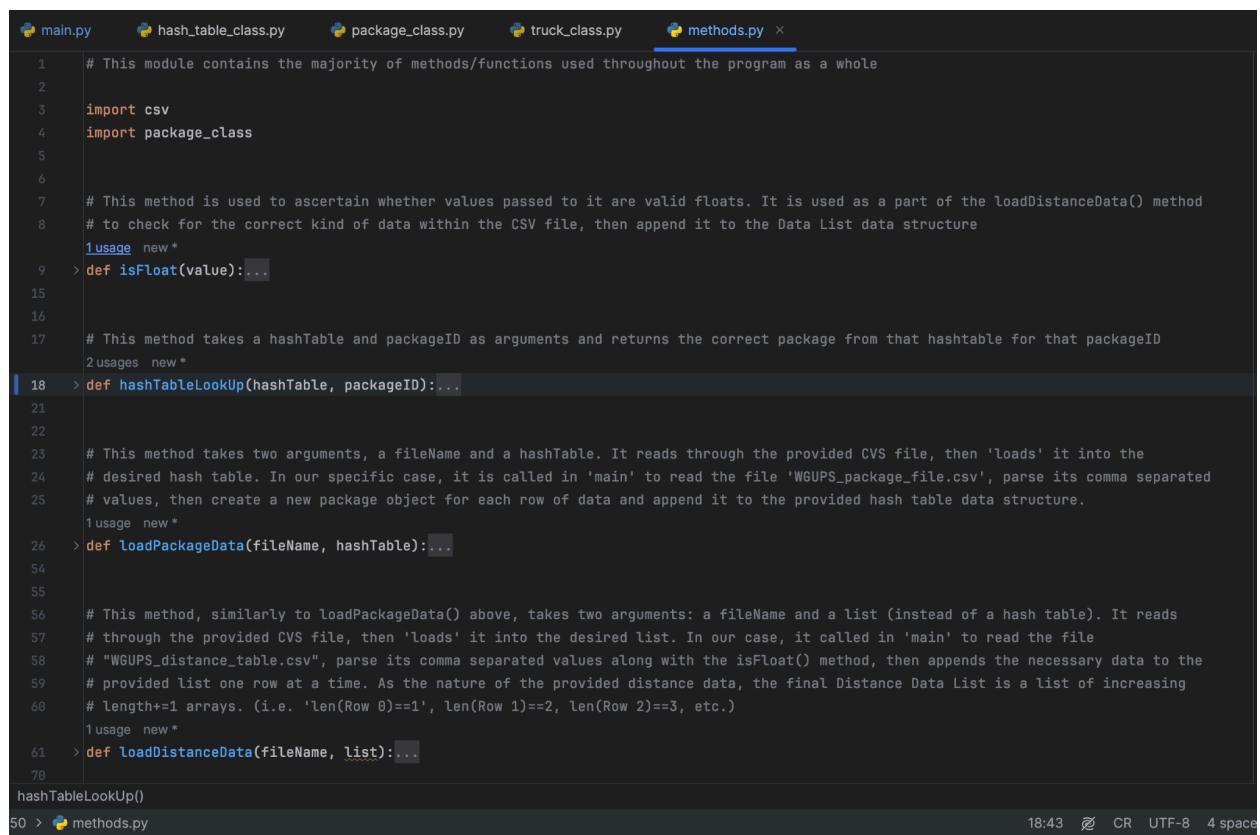
```
1 # This is the Package Class used to store data parsed from the provided "WGUPS_package_file.csv" file, and then st
2 # as individual instances of Package Objects.
3 # usage: & Aaron Tandem
4 class Package:
5     # Aaron Tandem
6     def __init__(self, ID, address, city, state, zip, deadline, weight, status):
7         self.ID = ID
8         self.address = address
9         self.city = city
10        self.state = state
11        self.zip = zip
12        self.deadline = deadline
13        self.weight = weight
14        self.status = status
15        self.timeDelivered = "N/A"
16
17    # This is used to explicit format how package data is printed to the console
18    # Aaron Tandem
19    def __str__(self):
20        return "%s, %s, %s, %s, %s, %s, %s" % (self.ID, self.address, self.city, self.state,
21                                                self.zip, self.deadline, self.weight, self.status, self.timeDelivered)
22
23    # Below are various getters and setters for members of this class
24
25    # 4 usages (4 dynamic) & Aaron Tandem
26    def getPackageID(self):
27        return self.ID
28
29    # 6 usages (6 dynamic) & Aaron Tandem
30    def getAddress(self):
31        return self.address
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
279
280
281
282
283
284
285
286
287
288
289
289
290
291
292
293
294
295
296
297
298
299
299
300
301
302
303
304
305
306
307
308
309
309
310
311
312
313
314
315
316
317
318
319
319
320
321
322
323
324
325
326
327
328
329
329
330
331
332
333
334
335
336
337
338
339
339
340
341
342
343
344
345
346
347
348
349
349
350
351
352
353
354
355
356
357
358
359
359
360
361
362
363
364
365
366
367
368
369
369
370
371
372
373
374
375
376
377
378
379
379
380
381
382
383
384
385
386
387
388
389
389
390
391
392
393
394
395
396
397
398
399
399
400
401
402
403
404
405
406
407
408
409
409
410
411
412
413
414
415
416
417
418
419
419
420
421
422
423
424
425
426
427
428
429
429
430
431
432
433
434
435
436
437
438
439
439
440
441
442
443
444
445
446
447
448
449
449
450
451
452
453
454
455
456
457
458
459
459
460
461
462
463
464
465
466
467
468
469
469
470
471
472
473
474
475
476
477
478
479
479
480
481
482
483
484
485
486
487
488
489
489
490
491
492
493
494
495
496
497
498
499
499
500
501
502
503
504
505
506
507
508
509
509
510
511
512
513
514
515
516
517
518
519
519
520
521
522
523
524
525
526
527
528
529
529
530
531
532
533
534
535
536
537
538
539
539
540
541
542
543
544
545
546
547
548
549
549
550
551
552
553
554
555
556
557
558
559
559
560
561
562
563
564
565
566
567
568
569
569
570
571
572
573
574
575
576
577
578
579
579
580
581
582
583
584
585
586
587
588
589
589
590
591
592
593
594
595
596
597
598
599
599
600
601
602
603
604
605
606
607
608
609
609
610
611
612
613
614
615
616
617
618
619
619
620
621
622
623
624
625
626
627
628
629
629
630
631
632
633
634
635
636
637
638
639
639
640
641
642
643
644
645
646
647
648
649
649
650
651
652
653
654
655
656
657
658
659
659
660
661
662
663
664
665
666
667
668
669
669
670
671
672
673
674
675
676
677
678
679
679
680
681
682
683
684
685
686
687
688
689
689
690
691
692
693
694
695
696
697
697
698
699
699
700
701
702
703
704
705
706
707
708
709
709
710
711
712
713
714
715
716
717
718
719
719
720
721
722
723
724
725
726
727
728
729
729
730
731
732
733
734
735
736
737
738
739
739
740
741
742
743
744
745
746
747
748
749
749
750
751
752
753
754
755
756
757
758
759
759
760
761
762
763
764
765
766
767
768
769
769
770
771
772
773
774
775
776
777
778
779
779
780
781
782
783
784
785
786
787
788
789
789
790
791
792
793
794
795
796
797
797
798
799
799
800
801
802
803
804
805
806
807
808
809
809
810
811
812
813
814
815
816
817
818
819
819
820
821
822
823
824
825
826
827
828
829
829
830
831
832
833
834
835
836
837
838
839
839
840
841
842
843
844
845
846
847
848
849
849
850
851
852
853
854
855
856
857
858
859
859
860
861
862
863
864
865
866
867
868
869
869
870
871
872
873
874
875
876
877
878
879
879
880
881
882
883
884
885
886
887
888
889
889
890
891
892
893
894
895
896
897
897
898
899
899
900
901
902
903
904
905
906
907
908
909
909
910
911
912
913
914
915
916
917
918
919
919
920
921
922
923
924
925
926
927
928
929
929
930
931
932
933
934
935
936
937
938
939
939
940
941
942
943
944
945
946
947
948
948
949
950
951
952
953
954
955
956
957
958
959
959
960
961
962
963
964
965
966
967
968
969
969
970
971
972
973
974
975
976
977
978
979
979
980
981
982
983
984
985
986
987
987
988
989
989
990
991
992
993
994
995
996
997
998
999
999
1000
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1048
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1068
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1078
1079
1080
1081
1082
1083
1084
1085
1086
1087
1087
1088
1089
1089
1090
1091
1092
1093
1094
1095
1096
1096
1097
1098
1099
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1138
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148
1148
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1168
1169
1170
1171
1172
1173
1174
1175
1176
1177
1178
1178
1179
1180
1181
1182
1183
1184
1185
1186
1187
1187
1188
1189
1190
1191
1192
1193
1194
1195
1195
1196
1197
1198
1199
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1218
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1238
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1248
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1259
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1268
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1278
1279
1280
1281
1282
1283
1284
1285
1286
1287
1287
1288
1289
1290
1291
1292
1293
1294
1295
1296
1296
1297
1298
1299
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1318
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1338
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1348
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1368
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1378
1379
1380
1381
1382
1383
1384
1385
1386
1387
1387
1388
1389
1390
1391
1392
1393
1394
1395
1396
1396
1397
1398
1399
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1418
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1428
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1438
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1448
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1458
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1468
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1478
1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1488
1489
1490
1491
1492
1493
1494
1495
1496
1496
1497
1498
1499
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1518
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1528
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1538
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1548
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1568
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1578
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1588
1589
1590
1591
1592
1593
1594
1595
1596
1596
1597
1598
1599
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1618
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1628
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1648
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1668
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1678
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1688
1689
1690
1691
1692
1693
1694
1695
1696
1696
1697
1698
1699
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1718
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1728
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1738
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1748
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1768
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1778
1779
1780
1781
1782
1783
1784
1785
1786
1787
1788
1788
1789
1790
1791
1792
1793
1794
1795
1796
1796
1797
1798
1799
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1818
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1828
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1838
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1848
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1868
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1878
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1888
1889
1890
1891
1892
1893
1894
1895
1896
1896
1897
1898
1899
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1918
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1928
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1938
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1948
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1968
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2018
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2038
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2048
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2068
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2078
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2088
2089
2090
2091
2092
2093
2094
2095
2096
2097
2098
2098
2099
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2109
2109
2110
2111
2112
2113
2114
2115
2116
2117
2118
2118
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2129
2129
2130
2131
2132
2133
2134
2135
2136
2137
2138
2138
2139
2140
2141
2142
2143
2144
2145
2146
2147
2148
2148
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2159
2159
2160
2161
2162
2163
2164
2165
2166
2167
2168
2168
2169
2170
2171
2172
2173
2174
2175
2176
2177
2178
2178
2179
2180
2181
2182
2183
2184
2185
2186
2187
2188
2188
2189
2190
2191
2192
2193
2194
2195
2196
2197
2198
2198
2199
2199
2200
2201
220
```

truck_class.py:

```

main.py          hash_table_class.py    package_class.py    truck_class.py    methods.py
1  # This is the Package Class used to store a list of Package Objects added to it from the hash table. It contains various methods for
2  # interacting with its members and manipulating Package Objects during the delivery process from 'main'.
3 usages  ± Aaron Tandem
4 class Truck:
5     ± Aaron Tandem
6     def __init__(self, truckNumber):
7         self.packages = []
8         self.truckNumber = truckNumber
9         self.startingAddress = "4001 South 700 East, Salt Lake City, UT 84107"
10        self.totalDeliveryDistance = 0
11        self.startTime = None
12        self.endTime = None
13
14    # This method will either append a package to the packages list or print that the list is full
15    40 usages  ± Aaron Tandem
16    def addPackage(self, package):
17        if len(self.packages) < 16:
18            self.packages.append(package)
19        else:
20            print("Truck is full for Truck #" + str(self.truckNumber) + ". No more packages can be added.")
21
22    # This method simply removes a package from the packages list
23    ± Aaron Tandem
24    def removePackage(self, packageID):
25        for package in self.packages:
26            if package.getPackageID() == packageID:
27                self.packages.remove(package)
28
29    ? # This method is similar to removePackage() but it also updates the hash table to say "delivered" along with the delivery time. It is
30    # called as a part of the deliverTruckPackages() method in 'main' and is a vital component of the Nearest Neighbor algorithm's logic.
31    1 usage (dynamic)  ± Aaron Tandem
32    def dropOffPackage(self, packageID, hashTable, deliveryTime):
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
Truck
50 > truck_class.py
26:138  LF  UTF-8  4 spa

```

methods.py:


```

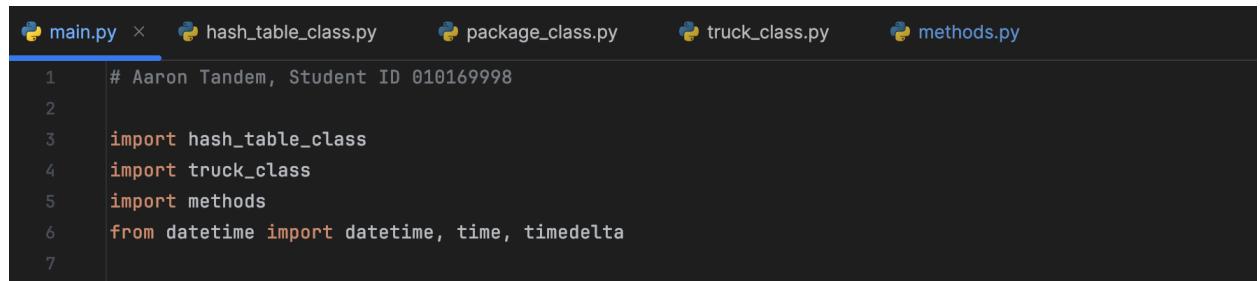
1 # This module contains the majority of methods/functions used throughout the program as a whole
2
3 import csv
4 import package_class
5
6
7 # This method is used to ascertain whether values passed to it are valid floats. It is used as a part of the loadDistanceData() method
8 # to check for the correct kind of data within the CSV file, then append it to the Data List data structure
9 1usage new*
9 > def isFloat(value):...
15
16
17 # This method takes a hashTable and packageID as arguments and returns the correct package from that hashtable for that packageID
17 2usages new*
18 > def hashTableLookUp(hashTable, packageID):...
21
22
23 # This method takes two arguments, a fileName and a hashTable. It reads through the provided CVS file, then 'loads' it into the
24 # desired hash table. In our specific case, it is called in 'main' to read the file 'WGUPS_package_file.csv', parse its comma separated
25 # values, then create a new package object for each row of data and append it to the provided hash table data structure.
25 1usage new*
26 > def loadPackageData(fileName, hashTable):...
27
28
29 # This method, similarly to loadPackageData() above, takes two arguments: a fileName and a list (instead of a hash table). It reads
30 # through the provided CVS file, then 'Loads' it into the desired list. In our case, it called in 'main' to read the file
31 # "WGUPS_distance_table.csv", parse its comma separated values along with the isFloat() method, then appends the necessary data to the
32 # provided list one row at a time. As the nature of the provided distance data, the final Distance Data List is a list of increasing
33 # length+=1 arrays. (i.e. 'len(Row 0)==1', len(Row 1)==2, len(Row 2)==3, etc.)
33 1usage new*
34 > def loadDistanceData(fileName, list):...
35
36 hashTableLookUp()
37
38 > methods.py

```

18:43 CR UTF-8 4 space

C1. Identification Information

Main.py screenshot showing StudentID:



```

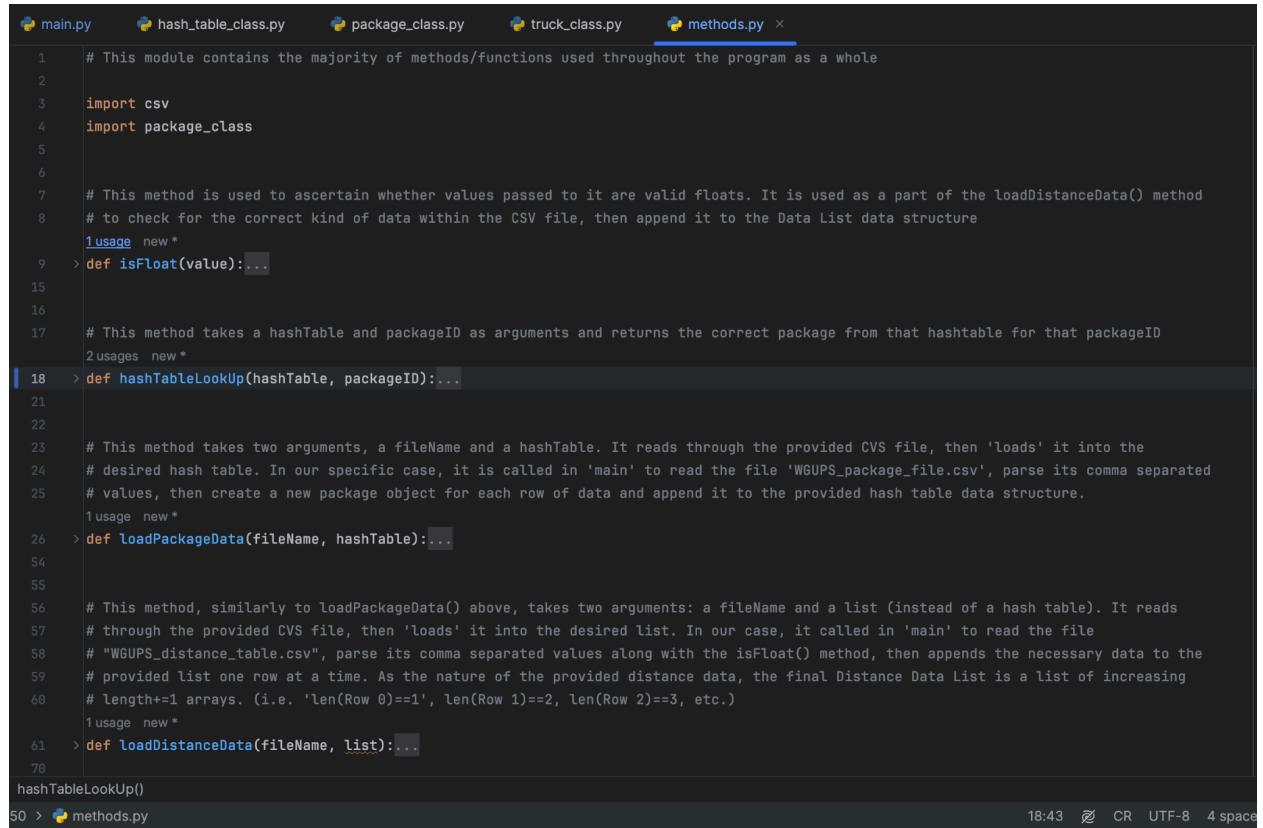
1 # Aaron Tandem, Student ID 010169998
2
3 import hash_table_class
4 import truck_class
5 import methods
6 from datetime import datetime, time, timedelta
7

```

C2. Process and Flow Comments

Below are some screenshots showing detailed comments explaining the processes and flow of the program for WGUPS:

C950 Task - 2 WGUPS Write-Up



```
1 # This module contains the majority of methods/functions used throughout the program as a whole
2
3 import csv
4 import package_class
5
6
7 # This method is used to ascertain whether values passed to it are valid floats. It is used as a part of the loadDistanceData() method
8 # to check for the correct kind of data within the CSV file, then append it to the Data List data structure
9 usage new*
10 > def isFloat(value):...
11
12
13 # This method takes a hashTable and packageID as arguments and returns the correct package from that hashtable for that packageID
14 usages new*
15 > def hashTableLookUp(hashTable, packageID):...
16
17
18 # This method takes two arguments, a fileName and a hashTable. It reads through the provided CVS file, then 'loads' it into the
19 # desired hash table. In our specific case, it is called in 'main' to read the file 'WGUPS_package_file.csv', parse its comma separated
20 # values, then create a new package object for each row of data and append it to the provided hash table data structure.
21 usage new*
22 > def loadPackageData(fileName, hashTable):...
23
24
25 # This method, similarly to loadPackageData() above, takes two arguments: a fileName and a list (instead of a hash table). It reads
26 # through the provided CVS file, then 'Loads' it into the desired list. In our case, it called in 'main' to read the file
27 # "WGUPS_distance_table.csv", parse its comma separated values along with the isFloat() method, then appends the necessary data to the
28 # provided list one row at a time. As the nature of the provided distance data, the final Distance Data List is a list of increasing
29 # length+1 arrays. (i.e. 'len(Row 0)==1', len(Row 1)==2, len(Row 2)==3, etc.)
30 usage new*
31 > def loadDistanceData(fileName, list):...
32
33 hashTableLookUp()
34 > methods.py 18:43 CR UTF-8 4 space
```

```
# Truck 2 will prioritize LATE on arrival packages that still have an early deadline
# NOTE: IT WILL LEAVE @ 9:10am!
truck2 = truck_class.Truck(2)
# packages 3, 18, 36, 38 must all be on Truck #2
truck2.addPackage(packageHashTable.search(3))
truck2.addPackage(packageHashTable.search(18))
truck2.addPackage(packageHashTable.search(36))
truck2.addPackage(packageHashTable.search(38))
# Truck 2 PRIORITY packages:
truck2.addPackage(packageHashTable.search(6))
truck2.addPackage(packageHashTable.search(25))
truck2.addPackage(packageHashTable.search(33))
truck2.addPackage(packageHashTable.search(12))
truck2.addPackage(packageHashTable.search(23))
truck2.addPackage(packageHashTable.search(24))
truck2.addPackage(packageHashTable.search(26))
truck2.addPackage(packageHashTable.search(27))
```

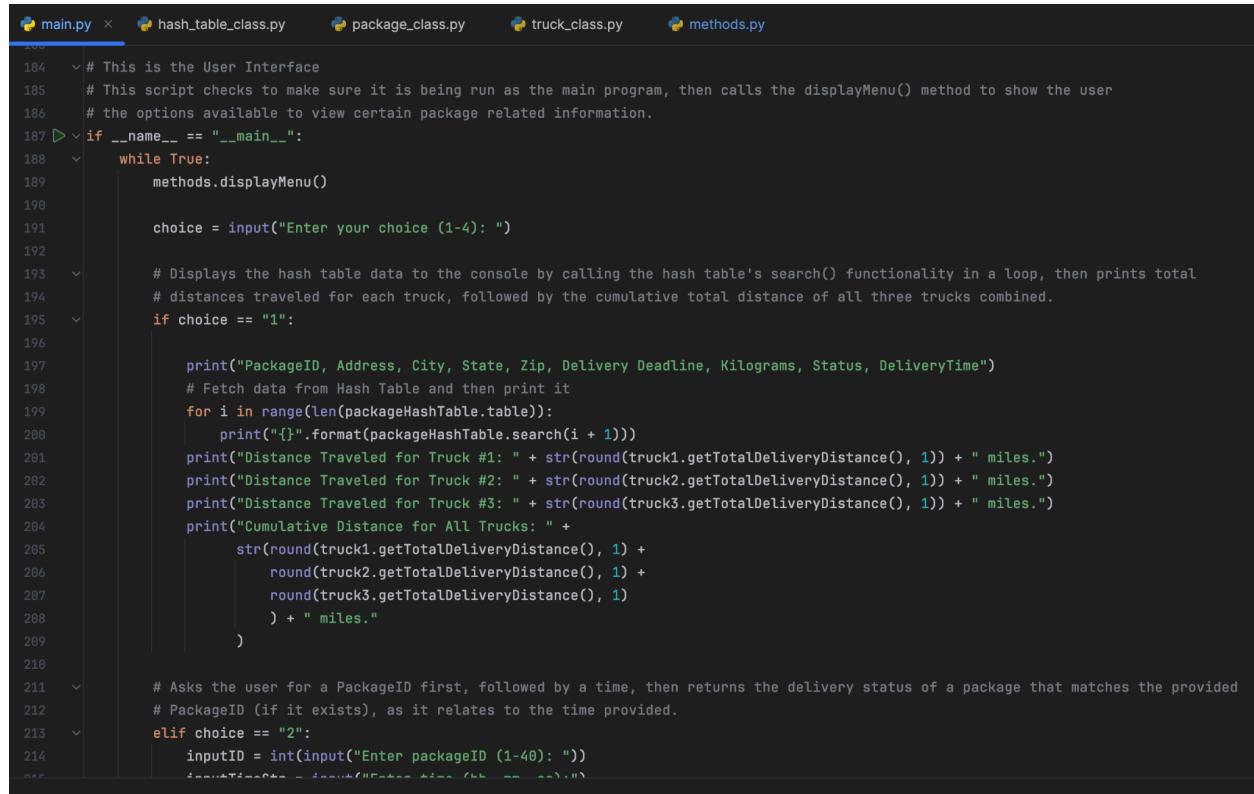
C950 Task - 2 WGUPS Write-Up

```
# This method finds and returns how many miles there are between two provided addresses held within a list storing address data.  
# It is subsequently used in the minDistanceFrom() method below to sequentially find which two addresses are closest to one another.  
4 usages ▲ Aaron Tandem  
def distanceBetween(address1, address2):...  
  
# This method makes use of the distanceBetween() function above to sequentially find the address of a package in a truck's list that is  
# closest to the provided 'fromAddress' (i.e. 'where the truck was last located'). This 'fromAddress' generally always starts at the WGUPS  
# HUB located at "4001 South 700 East, Salt Lake City, UT 84107". However, this address is automatically updated to the address of the last  
# delivered package in the recursive call of minDistanceFrom() in the deliverTruckPackages() method below. This is a critical part of the  
# Nearest Neighbor Greedy Algorithm used by this program to solve WGUPS's package delivery efficiency issues.  
1 usage ▲ Aaron Tandem  
def minDistanceFrom(fromAddress, truckPackages):...  
  
# This method is the final part of the Nearest Neighbor algorithm chosen for this project, and the part of the algorithm that  
# simultaneously removes (i.e. "delivers") a package from a truck's package list and update's that truck's "startingAddress" as the same  
# address of the package that was just removed/delivered.  
# The second argument "startTime" must be of format 'time(x, x, x)', including when calling getStartingTime() or getEndingTime() from the  
# provided 'truck' argument.  
3 usages ▲ Aaron Tandem  
def deliverTruckPackages(truck, startTime):...
```

D. Interface

Below are screenshots showing the code and functionality of the user interface:

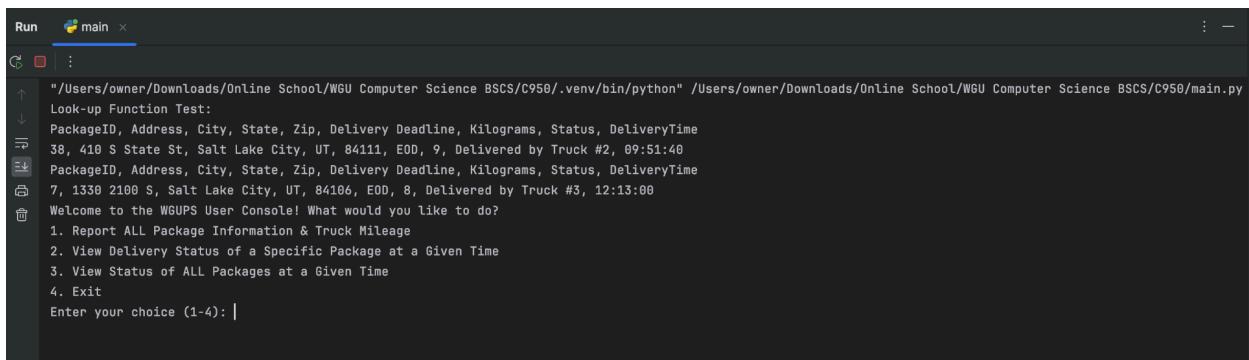
Code: (can be viewed in greater detail in the provided zip file)



```
main.py x hash_table_class.py package_class.py truck_class.py methods.py  
184 # This is the User Interface  
185 # This script checks to make sure it is being run as the main program, then calls the displayMenu() method to show the user  
186 # the options available to view certain package related information.  
187 if __name__ == "__main__":  
188     while True:  
189         methods.displayMenu()  
190  
191         choice = input("Enter your choice (1-4): ")  
192  
193         # Displays the hash table data to the console by calling the hash table's search() functionality in a loop, then prints total  
194         # distances traveled for each truck, followed by the cumulative total distance of all three trucks combined.  
195         if choice == "1":  
196             print("PackageID, Address, City, State, Zip, Delivery Deadline, Kilograms, Status, DeliveryTime")  
197             # Fetch data from Hash Table and then print it  
198             for i in range(len(packageHashTable.table)):  
199                 print("{}{}".format(packageHashTable.search(i + 1)))  
200             print("Distance Traveled for Truck #1: " + str(round(truck1.getTotalDeliveryDistance(), 1)) + " miles.")  
201             print("Distance Traveled for Truck #2: " + str(round(truck2.getTotalDeliveryDistance(), 1)) + " miles.")  
202             print("Distance Traveled for Truck #3: " + str(round(truck3.getTotalDeliveryDistance(), 1)) + " miles.")  
203             print("Cumulative Distance for All Trucks: " +  
204                 str(round(truck1.getTotalDeliveryDistance(), 1) +  
205                     round(truck2.getTotalDeliveryDistance(), 1) +  
206                     round(truck3.getTotalDeliveryDistance(), 1)  
207                 ) + " miles."  
208             )  
209  
210         # Asks the user for a PackageID first, followed by a time, then returns the delivery status of a package that matches the provided  
211         # PackageID (if it exists), as it relates to the time provided.  
212         elif choice == "2":  
213             inputID = int(input("Enter packageID (1-40): "))
```

C950 Task - 2 WGUPS Write-Up

CLI interface:



The screenshot shows a terminal window titled "Run main". The terminal displays the following output:

```
"C:\Users\owner\Downloads\Online School\WGU Computer Science BSCS\C950\.venv\bin\python" /Users\owner\Downloads\Online School\WGU Computer Science BSCS\C950/main.py
Look-up Function Test:
PackageID, Address, City, State, Zip, Delivery Deadline, Kilograms, Status, DeliveryTime
38, 410 S State St, Salt Lake City, UT, 84111, EOD, 9, Delivered by Truck #2, 09:51:40
PackageID, Address, City, State, Zip, Delivery Deadline, Kilograms, Status, DeliveryTime
7, 1330 2100 S, Salt Lake City, UT, 84106, EOD, 8, Delivered by Truck #3, 12:13:00
Welcome to the WGUPS User Console! What would you like to do?
1. Report ALL Package Information & Truck Mileage
2. View Delivery Status of a Specific Package at a Given Time
3. View Status of ALL Packages at a Given Time
4. Exit
Enter your choice (1-4): |
```

D1. First Status Check

Screenshot of the first status check at 9:00am:

```
Delivery Status for All Packages as of 09:00:00:  
Package 1 was Delivered by Truck #1 at 08:39:00 to 195 W Oakland Ave.  
Package 2 is en route as of 09:00:00.  
Package 3 is en route as of 09:00:00.  
Package 4 is en route as of 09:00:00.  
Package 5 is en route as of 09:00:00.  
Package 6 is en route as of 09:00:00.  
Package 7 is en route as of 09:00:00.  
Package 8 is en route as of 09:00:00.  
Package 9 is en route as of 09:00:00.  
Package 10 is en route as of 09:00:00.  
Package 11 is en route as of 09:00:00.  
Package 12 is en route as of 09:00:00.  
Package 13 is en route as of 09:00:00.  
Package 14 was Delivered by Truck #1 at 08:06:20 to 4300 S 1300 E.  
Package 15 was Delivered by Truck #1 at 08:13:00 to 4580 S 2300 E.  
Package 16 was Delivered by Truck #1 at 08:13:00 to 4580 S 2300 E.  
Package 17 is en route as of 09:00:00.  
Package 18 is en route as of 09:00:00.  
Package 19 is en route as of 09:00:00.  
Package 20 was Delivered by Truck #1 at 08:48:00 to 3595 Main St.  
Package 21 is en route as of 09:00:00.  
Package 22 is en route as of 09:00:00.  
Package 23 is en route as of 09:00:00.  
Package 24 is en route as of 09:00:00.  
Package 25 is en route as of 09:00:00.  
Package 26 is en route as of 09:00:00.  
Package 27 is en route as of 09:00:00.  
Package 28 is en route as of 09:00:00.  
Package 29 was Delivered by Truck #1 at 08:29:40 to 1330 2100 S.
```

```
Package 30 is en route as of 09:00:00.  
Package 31 was Delivered by Truck #1 at 08:53:20 to 3365 S 900 W.  
Package 32 is en route as of 09:00:00.  
Package 33 is en route as of 09:00:00.  
Package 34 was Delivered by Truck #1 at 08:13:00 to 4580 S 2300 E.  
Package 35 is en route as of 09:00:00.  
Package 36 is en route as of 09:00:00.  
Package 37 is en route as of 09:00:00.  
Package 38 is en route as of 09:00:00.  
Package 39 is en route as of 09:00:00.  
Package 40 was Delivered by Truck #1 at 08:42:40 to 380 W 2880 S.
```

D2. Second Status Check

Screenshot of the second status check at 10:05am:

```
Delivery Status for All Packages as of 10:05:00:  
Package 1 was Delivered by Truck #1 at 08:39:00 to 195 W Oakland Ave.  
Package 2 is en route as of 10:05:00.  
Package 3 was Delivered by Truck #2 at 09:55:00 to 233 Canyon Rd.  
Package 4 is en route as of 10:05:00.  
Package 5 is en route as of 10:05:00.  
Package 6 is en route as of 10:05:00.  
Package 7 is en route as of 10:05:00.  
Package 8 is en route as of 10:05:00.  
Package 9 is en route as of 10:05:00.  
Package 10 is en route as of 10:05:00.  
Package 11 is en route as of 10:05:00.  
Package 12 is en route as of 10:05:00.  
Package 13 was Delivered by Truck #1 at 09:12:40 to 2010 W 500 S.  
Package 14 was Delivered by Truck #1 at 08:06:20 to 4300 S 1300 E.  
Package 15 was Delivered by Truck #1 at 08:13:00 to 4580 S 2300 E.  
Package 16 was Delivered by Truck #1 at 08:13:00 to 4580 S 2300 E.  
Package 17 is en route as of 10:05:00.  
Package 18 is en route as of 10:05:00.  
Package 19 is en route as of 10:05:00.  
Package 20 was Delivered by Truck #1 at 08:48:00 to 3595 Main St.  
Package 21 is en route as of 10:05:00.  
Package 22 is en route as of 10:05:00.  
Package 23 is en route as of 10:05:00.  
Package 24 was Delivered by Truck #2 at 09:23:40 to 5025 State St.  
Package 25 was Delivered by Truck #2 at 09:18:00 to 5383 South 900 East #104.  
Package 26 was Delivered by Truck #2 at 09:18:00 to 5383 South 900 East #104.  
Package 27 is en route as of 10:05:00.  
Package 28 is en route as of 10:05:00.  
Package 29 was Delivered by Truck #1 at 08:29:40 to 1330 2100 S.
```



D3. Third Status Check

Screenshot of the third status check at 1:00pm:

```
Delivery Status for All Packages as of 13:00:00:  
Package 1 was Delivered by Truck #1 at 08:39:00 to 195 W Oakland Ave.  
Package 2 was Delivered by Truck #3 at 12:07:40 to 2530 S 500 E.  
Package 3 was Delivered by Truck #2 at 09:55:00 to 233 Canyon Rd.  
Package 4 was Delivered by Truck #3 at 11:47:40 to 380 W 2880 S.  
Package 5 was Delivered by Truck #3 at 12:28:20 to 410 S State St.  
Package 6 was Delivered by Truck #2 at 10:25:40 to 3060 Lester St.  
Package 7 was Delivered by Truck #3 at 12:13:00 to 1330 2100 S.  
Package 8 was Delivered by Truck #3 at 12:31:40 to 300 State St.  
Package 9 was Delivered by Truck #3 at 12:28:20 to 410 S State St.  
Package 10 was Delivered by Truck #3 at 12:22:20 to 600 E 900 South.  
Package 11 is en route as of 13:00:00.  
Package 12 was Delivered by Truck #2 at 11:05:40 to 3575 W Valley Central Station bus Loop.  
Package 13 was Delivered by Truck #1 at 09:12:40 to 2010 W 500 S.  
Package 14 was Delivered by Truck #1 at 08:06:20 to 4300 S 1300 E.  
Package 15 was Delivered by Truck #1 at 08:13:00 to 4580 S 2300 E.  
Package 16 was Delivered by Truck #1 at 08:13:00 to 4580 S 2300 E.  
Package 17 was Delivered by Truck #3 at 11:55:20 to 3148 S 1100 W.  
Package 18 was Delivered by Truck #2 at 10:38:40 to 1488 4800 S.  
Package 19 was Delivered by Truck #3 at 11:39:20 to 177 W Price Ave.  
Package 20 was Delivered by Truck #1 at 08:48:00 to 3595 Main St.  
Package 21 was Delivered by Truck #3 at 11:37:40 to 3595 Main St.  
Package 22 is en route as of 13:00:00.  
Package 23 was Delivered by Truck #2 at 10:40:40 to 5100 South 2700 West.  
Package 24 was Delivered by Truck #2 at 09:23:40 to 5025 State St.  
Package 25 was Delivered by Truck #2 at 09:18:00 to 5383 South 900 East #104.  
Package 26 was Delivered by Truck #2 at 09:18:00 to 5383 South 900 East #104.  
Package 27 was Delivered by Truck #2 at 10:11:00 to 1060 Dalton Ave S.  
Package 28 was Delivered by Truck #3 at 11:44:20 to 2835 Main St.  
Package 29 was Delivered by Truck #1 at 08:29:40 to 1330 2100 S.
```

 main.py

```
Package 30 was Delivered by Truck #1 at 09:26:40 to 300 State St.  
Package 31 was Delivered by Truck #1 at 08:53:20 to 3365 S 900 W.  
Package 32 was Delivered by Truck #3 at 11:53:20 to 3365 S 900 W.  
Package 33 was Delivered by Truck #2 at 09:38:00 to 2530 S 500 E.  
Package 34 was Delivered by Truck #1 at 08:13:00 to 4580 S 2300 E.  
Package 35 was Delivered by Truck #3 at 12:51:00 to 1060 Dalton Ave S.  
Package 36 was Delivered by Truck #2 at 10:20:20 to 2300 Parkway Blvd.  
Package 37 was Delivered by Truck #1 at 09:23:20 to 410 S State St.  
Package 38 was Delivered by Truck #2 at 09:51:40 to 410 S State St.  
Package 39 was Delivered by Truck #3 at 12:45:40 to 2010 W 500 S.  
Package 40 was Delivered by Truck #1 at 08:42:40 to 380 W 2880 S.
```

E. Screenshot of Code Execution

Below is a screenshot showing the successful completion of the code, including the total mileage traveled by all trucks:

```
PackageID, Address, City, State, Zip, Delivery Deadline, Kilograms, Status, DeliveryTime
1, 195 W Oakland Ave, Salt Lake City, UT, 84115, 10:30 AM, 21, Delivered by Truck #1, 08:39:00
2, 2530 S 500 E, Salt Lake City, UT, 84106, EOD, 44, Delivered by Truck #3, 12:07:40
3, 233 Canyon Rd, Salt Lake City, UT, 84103, EOD, 2, Delivered by Truck #2, 09:55:00
4, 380 W 2880 S, Salt Lake City, UT, 84115, EOD, 4, Delivered by Truck #3, 11:47:40
5, 410 S State St, Salt Lake City, UT, 84111, EOD, 5, Delivered by Truck #3, 12:28:20
6, 3060 Lester St, West Valley City, UT, 84119, 10:30 AM, 88, Delivered by Truck #2, 10:25:40
7, 1330 2100 S, Salt Lake City, UT, 84106, EOD, 8, Delivered by Truck #3, 12:13:00
8, 300 State St, Salt Lake City, UT, 84103, EOD, 9, Delivered by Truck #3, 12:31:40
9, 410 S State St, Salt Lake City, UT, 84103, EOD, 2, Delivered by Truck #3, 12:28:20
10, 600 E 900 South, Salt Lake City, UT, 84105, EOD, 1, Delivered by Truck #3, 12:22:20
11, 2600 Taylorsville Blvd, Salt Lake City, UT, 84118, EOD, 1, Delivered by Truck #3, 13:15:20
12, 3575 W Valley Central Station bus Loop, West Valley City, UT, 84119, EOD, 1, Delivered by Truck #2, 11:05:40
13, 2010 W 500 S, Salt Lake City, UT, 84104, 10:30 AM, 2, Delivered by Truck #1, 09:12:40
14, 4300 S 1300 E, Millcreek, UT, 84117, 10:30 AM, 88, Delivered by Truck #1, 08:06:20
15, 4580 S 2300 E, Holladay, UT, 84117, 9:00 AM, 4, Delivered by Truck #1, 08:13:00
16, 4580 S 2300 E, Holladay, UT, 84117, 10:30 AM, 88, Delivered by Truck #1, 08:13:00
17, 3148 S 1100 W, Salt Lake City, UT, 84119, EOD, 2, Delivered by Truck #3, 11:55:20
18, 1488 4800 S, Salt Lake City, UT, 84123, EOD, 6, Delivered by Truck #2, 10:38:40
19, 177 W Price Ave, Salt Lake City, UT, 84115, EOD, 37, Delivered by Truck #3, 11:39:20
20, 3595 Main St, Salt Lake City, UT, 84115, 10:30 AM, 37, Delivered by Truck #1, 08:48:00
21, 3595 Main St, Salt Lake City, UT, 84115, EOD, 3, Delivered by Truck #3, 11:37:40
22, 6351 South 900 East, Murray, UT, 84121, EOD, 2, Delivered by Truck #3, 13:38:00
23, 5100 South 2700 West, Salt Lake City, UT, 84118, EOD, 5, Delivered by Truck #2, 10:40:40
24, 5025 State St, Murray, UT, 84107, EOD, 7, Delivered by Truck #2, 09:23:40
25, 5383 South 900 East #104, Salt Lake City, UT, 84117, 10:30 AM, 7, Delivered by Truck #2, 09:18:00
26, 5383 South 900 East #104, Salt Lake City, UT, 84117, EOD, 25, Delivered by Truck #2, 09:18:00
27, 1060 Dalton Ave S, Salt Lake City, UT, 84104, EOD, 5, Delivered by Truck #2, 10:11:00
28, 2835 Main St, Salt Lake City, UT, 84115, EOD, 7, Delivered by Truck #3, 11:44:20
29, 1330 2100 S, Salt Lake City, UT, 84106, 10:30 AM, 2, Delivered by Truck #1, 08:29:40
```

```
30, 300 State St, Salt Lake City, UT, 84103, 10:30 AM, 1, Delivered by Truck #1, 09:26:40
31, 3365 S 900 W, Salt Lake City, UT, 84119, 10:30 AM, 1, Delivered by Truck #1, 08:53:20
32, 3365 S 900 W, Salt Lake City, UT, 84119, EOD, 1, Delivered by Truck #3, 11:53:20
33, 2530 S 500 E, Salt Lake City, UT, 84106, EOD, 1, Delivered by Truck #2, 09:38:00
34, 4580 S 2300 E, Holladay, UT, 84117, 10:30 AM, 2, Delivered by Truck #1, 08:13:00
35, 1060 Dalton Ave S, Salt Lake City, UT, 84104, EOD, 88, Delivered by Truck #3, 12:51:00
36, 2300 Parkway Blvd, West Valley City, UT, 84119, EOD, 88, Delivered by Truck #2, 10:20:20
37, 410 S State St, Salt Lake City, UT, 84111, 10:30 AM, 2, Delivered by Truck #1, 09:23:20
38, 410 S State St, Salt Lake City, UT, 84111, EOD, 9, Delivered by Truck #2, 09:51:40
39, 2010 W 500 S, Salt Lake City, UT, 84104, EOD, 9, Delivered by Truck #3, 12:45:40
40, 380 W 2880 S, Salt Lake City, UT, 84115, 10:30 AM, 45, Delivered by Truck #1, 08:42:40
Distance Traveled for Truck #1: 33.6 miles.
Distance Traveled for Truck #2: 42.3 miles.
Distance Traveled for Truck #3: 41.7 miles.
Cumulative Distance for All Trucks: 117.6000000000001 miles.
```

F1. Strengths of the Chosen Algorithm

The algorithm chosen for this project was the Nearest Neighbor Algorithm. In simple terms, this algorithm is greedy in nature, meaning that it chooses the option that is “best now” (i.e. “in the short-term”) to achieve an optimal, if not perfectly ideal, solution. This algorithm’s strengths include:

- Being self-adjusting in that its implementation is not dependent on a specific data set, but can account for increasing, decreasing, or completely different data points
- A time-complexity of only $O(N^2)$ when implemented correctly
- It is relatively easy to implement, and does not require extensively detailed knowledge in the field of computer science or heuristics

F2. Verification of Algorithm

The implementation of the Nearest Neighbor Algorithm in this project was able to keep the total cumulative miles for all three trucks less than 120 miles. Additionally, all packages were delivered on time as required. With further optimization, even less mileage, with potential for quicker delivery times, could be achieved.

F3. Other Possible Algorithms

Two other algorithms that could potentially replace the Nearest Neighbor implementation in this project are the 2-Opt Algorithm and Dijkstra’s Shortest Path. Both of these algorithms are concerned with finding an ideal path between points of “travel” by reducing total length of said travel.

F3a. Algorithm Differences

In simple terms, 2-Opt is similar to the Nearest Neighbor (and may even implement the Nearest Neighbor algorithm as part of its implementation), but differs in how it gets its name — by iteratively “swapping” pairs of edges in a route and seeing if that improves travel time and distance. Dijkstra’s Shortest Path on the other hand, systematically considers the entirety of paths between two points and chooses the shortest one, while Nearest Neighbor only makes local decisions based on neighboring nodes.

G. Different Approach

If I were to do this project again, one of the main things I would do is to create an algorithm that automatically loads the trucks in an optimal manner. This could be done relatively simply by implementing a version of the Nearest Neighbor algorithm that chooses which packages go best with others based on their delivery addresses. Next, from an organizational perspective, I would like to ensure that *all* methods not belonging to specific classes are kept in a specific file so that code neatness can be maintained to the utmost within the main program itself.

H. Verification of Data Structure

The hash table satisfies all requirements for this project and helps improve WGUPS's systems in a number of ways. Mainly, it is a fast and self-adjusting data structure. Meaning, all operations directly related to updating, inserting, removing, and searching with the hash table operate in $O(N)$ time. Additionally, the hash table can easily and automatically be increased or decreased in size in relation to changing data sets without sacrificing any of this speed.

H1. Other Data Structures

Two other data structures that could meet the same requirements for this scenario are lists and arrays. Both of these data structures are not inherently self-adjusting, but can be made to be so with extra steps during implementation.

H1a. Data Structure Differences

Lists differ from hash tables in that this data structure is linear in nature, meaning that all data is stored sequentially within them. Additionally, values are generally accessed via indices rather than via key-value mapping like in hash tables. Arrays on the other hand have a tendency of being fixed in size and only storing one data type. However, because we usually will know how many packages we'll be delivering, along with the fact that the array would only store package objects, it's possible it could be used. However, it is definitely not the ideal data structure to use as a solution to this scenario.

I. Sources

Lysecky, R., & Vahid, F. (2018, June). *C950: Data Structures and Algorithms II*. zyBooks.

Retrieved March 22, 2021, from

<https://learn.zybooks.com/zybook/WGUC950AY20182019/>