Kimberly Tom

CS 325

3/7/19

Homework 8

1) **a)**

```
# put each item as you come to it into the first (earliest opened) bin into which it fits
# if there is no available bin then open a new bin
# https://www.geeksforgeeks.org/bin-packing-problem-minimize-number-of-used-bins/
def firstFitAlgorithm(numItems, binCapacity, itemWeights):
    # numberOfBins holds the total number of bins we are using, start with 1 bin
    numberOfBins = 1

    # create an array totalBins to hold all the bins we are using
    totalBins = []

    # create a bin with max capacity
    makeBin = Bin(binCapacity)

    # add the created bin to totalBins array
    totalBins.append(makeBin)

    # place items into the bins
    for i in range(numItems):
        itemStored = 0

        # go through the bins
        for b in range(numberOfBins):
            # if item can fit into a bin we already created, store in that bin
            if itemWeights[i] <= totalBins[b].capacity:
                totalBins[b].capacity = totalBins[b].capacity - itemWeights[i]
                itemStored = 1
                break
         # if not, store in a new bin
        if not itemStored:
            newBinCapacity = binCapacity - itemWeights[i]
            makeBin = Bin(newBinCapacity)
            totalBins.append(makeBin)
            numberOfBins = numberOfBins + 1
```

```
    return numberOfBins



# first sort the items in decreasing order by size, then
use First-Fit on the resulting list
def decreasingAlgorithm(numItems, binCapacity,
itemWeights):
    # make a copy of the array itemWEights
    decreasingWeights = itemWeights.copy()

    # sort the array decreasingWeights from greatest to
least weight
    # https://www.geeksforgeeks.org/python-list-sort/
    decreasingWeights.sort(reverse = True)

    # call firstFitAlgorithm function and store in number
of bins then return the number of bins
    numberOfBins = firstFitAlgorithm(numItems, binCapacity,
decreasingWeights)

    return numberOfBins

# Place the items in order in which they arrive. Place next
item into bin which will leave the least room
# left over after the item is placed in the bin.  If it
doesn't fit in any bin, start new bin
def bestFitAlgorithm(numItems, binCapacity, itemWeights):
    # numberOfBins holds the total number of bins we are
using, start with 1 bin
    numberOfBins = 1

    # create an array totalBins to hold all the bins we are
using
    totalBins = []

    # create a bin with max capacity
    makeBin = Bin(binCapacity)

    # add the created bin to totalBins array
    totalBins.append(makeBin)

    # place items into the bins
    for i in range(numItems):
        tempStore = -1
        leastRoomLeftOver = binCapacity
        currentBinRoomLeftOver = 0
```

```
        # go through the bins
        for b in range(numberOfBins):
            # if item can fit into a bin we already
created, potentially store in that bin
            if itemWeights[i] <= totalBins[b].capacity:
                currentBinRoomLeftOver =
totalBins[b].capacity - itemWeights[i]
                # if the current bin has less room left
over than the prior least room left over bin, update temp
bin
                if currentBinRoomLeftOver <
leastRoomLeftOver:
                    leastRoomLeftOver =
currentBinRoomLeftOver
                    tempStore = totalBins[b]

        # if no bin has enough room, store in a new bin
        if tempStore == -1:
            newBinCapacity = binCapacity - itemWeights[i]
            makeBin = Bin(newBinCapacity)
            totalBins.append(makeBin)
            numberOfBins = numberOfBins + 1

        # placee the item in the bin with the least amount
of room leftover
        else:
            tempStore.capacity = tempStore.capacity -
itemWeights[i]

    return numberOfBins
```

Running Time for first-fit is O(n^2) because for each of the n items, we need to place the item in a bin so we need to run through the list n times for that. However, while going through each item, we need to run through the bins to see where the item fits. Therefore the running time can't be bigger than O(n^2), because there will be at most n bins if in the worst case, where each item is the same capacity as a bin. So for each item, we will need to look at at most n bins and therefore do work at most n times per item.

Running Time for First Fit Decreasing is O(n^2) because this function basically sorts the array, then calls the first-fit algorithm, which we said above was O(n^2). The first-fit algorithm dominates the sorting part of the First Fit Decreasing function because the python sort function takes O(nlogn) time to run, which is less than O(n^2). So, O(nlogn ^ n^2) simplifies to O(n^2).

Running Time for Best Fit is O(n^2) because for each of the n items, we need to place the item in a bin so we need to run through the list n times for that. Then, for each item, we run through all the bins we have created so far to see where we should place it in order to place it in a bin that will have the least space remaining. This would happen at most n times, because the worst case scenario would be when all n items are at max capacity, so we have to keep adding a new bin. So the last item to be placed will look through a maximum of n-1 bins to be placed.

b) see zip file in TEACH

c) Test Case # 1

First Fit: 92

First Fit Decreasing: 87

Best Fit: 91

Test Case # 2

First Fit: 113

First Fit Decreasing: 107

Best Fit: 112

Test Case # 3

First Fit: 245

First Fit Decreasing: 237

Best Fit: 243

Test Case # 4

First Fit: 190

First Fit Decreasing: 183

Best Fit: 188

Test Case # 5

First Fit: 296

First Fit Decreasing: 289

Best Fit: 291

Test Case # 6

First Fit: 526

First Fit Decreasing: 509

Best Fit: 519

Test Case # 7

First Fit: 107

First Fit Decreasing: 101

Best Fit: 105

Test Case # 8

First Fit: 361

First Fit Decreasing: 351

Best Fit: 358

Test Case # 9

First Fit: 76

First Fit Decreasing: 72

Best Fit: 75

Test Case # 10

First Fit: 417

First Fit Decreasing: 400

Best Fit: 412

Test Case # 11

First Fit: 316

First Fit Decreasing: 302

Best Fit: 312

Test Case # 12

First Fit: 155

First Fit Decreasing: 149

Best Fit: 152

Test Case # 13

First Fit: 24

First Fit Decreasing: 22

Best Fit: 24

Test Case # 14

First Fit: 53

First Fit Decreasing: 50

Best Fit: 52

Test Case # 15

First Fit: 60

First Fit Decreasing: 58

Best Fit: 60

Test Case # 16

First Fit: 232

First Fit Decreasing: 221

Best Fit: 231

Test Case # 17

First Fit: 246

First Fit Decreasing: 236

Best Fit: 242

Test Case # 18

First Fit: 140

First Fit Decreasing: 130

Best Fit: 138

Test Case # 19

First Fit: 34

First Fit Decreasing: 33

Best Fit: 34

Test Case # 20

First Fit: 442

First Fit Decreasing: 422

Best Fit: 438

The best algorithm is the first fit decreasing by far as it used the least amount of bins in all 20 cases. Second best algorithm is Best fit, and last is first fit. best fit and first fit are generally pretty close, but best fit performs better than first fit on 17/20 tests. The 3 tests where best fit did not perform better than first fit, it performed the same as first fit.

To generate the inputs, I create a write file where to the random inputs will be stored. I set the testcases to 20, then write that to the file. Then for each of the 20 test cases, I use the random function to randomly generate a number between 1 and 1000 and set the capacity to this number, then write that to the new file. Then I randomly generate a number between 1 and 1000 and set the item count to this number, then write that to the new file. Then I generate random numbers between 1 and the capacity and write those to the file, for each item, and write those to the file. This is repeated 20 times. The new file will then have the number of test cases on one line, then for the 20 cases, have the capacity, item count, and item weights. I then run binpack.py using my new text file to generate the test cases and bin amounts for each algorithm.

2) used LINDO for part A and B

a) Six items S = { 4, 4, 4, 6, 6, 6} and bin capacity of 10

interpretation of result: The minimum number of bins used is 3 bins. In one bin, was the first item and fourth item. In another bin, was the second item and the sixth item. In a third bin, was the third and fifth item.

input

MIN a1 + a2 + a3 + a4 + a5 + a6

ST

a1 + a2 + a3 + a4 + a5 + a6 > 0

4b01 + 4b02 + 4b03 + 6b04 + 6b05 + 6b06 - 10a1 <= 0

4b11 + 4b12 + 4b13 + 6b14 + 6b15 + 6b16 - 10a2 <= 0

4b21 + 4b22 + 4b23 + 6b24 + 6b25 + 6b26 - 10a3 <= 0

4b31 + 4b32 + 4b33 + 6b34 + 6b35 + 6b36 - 10a4 <= 0

4b41 + 4b42 + 4b43 + 6b44 + 6b45 + 6b46 - 10a5 <= 0

4b51 + 4b52 + 4b53 + 6b54 + 6b55 + 6b56 - 10a6 <= 0

b01 + b11 + b21 + b31 + b41 + b51 = 1

b02 + b12 + b22 + b32 + b42 + b52 = 1

b03 + b13 + b23 + b33 + b43 + b53 = 1

b04 + b14 + b24 + b34 + b44 + b54 = 1

b05 + b15 + b25 + b35 + b45 + b55 = 1

b06 + b16 + b26 + b36 + b46 + b56 = 1

END

INT a1
INT a2
INT a3
INT a4
INT a5
INT a6

INT b01

INT b11

INT b21

INT b31

INT b41

INT b51

INT b02

INT b12

INT b22

INT b32

INT b42

INT b52

INT b03

INT b13

INT b23

INT b33

INT b43

INT b53
INT b04

INT b14

INT b24

INT b34

INT b44

INT b54

INT b05

INT b15

INT b25

INT b35

INT b45

INT b55

INT b06

INT b16

INT b26

INT b36

INT b46

INT b56

output:

```
LP OPTIMUM FOUND AT STEP      4

 OBJECTIVE VALUE =   3.00000000



 NEW INTEGER SOLUTION OF    3.00000000     AT BRANCH      0 PIVOT      4

 RE-INSTALLING BEST SOLUTION...

        OBJECTIVE FUNCTION VALUE

        1)      3.000000

  VARIABLE        VALUE          REDUCED COST
        A1         1.000000          1.000000
```

A2	0.000000	1.000000
A3	0.000000	1.000000
A4	0.000000	1.000000
A5	1.000000	1.000000
A6	1.000000	1.000000
B01	0.000000	0.000000
B11	0.000000	0.000000
B21	0.000000	0.000000
B31	0.000000	0.000000
B41	1.000000	0.000000
B51	0.000000	0.000000
B02	1.000000	0.000000
B12	0.000000	0.000000
B22	0.000000	0.000000
B32	0.000000	0.000000
B42	0.000000	0.000000
B52	0.000000	0.000000
B03	0.000000	0.000000
B13	0.000000	0.000000
B23	0.000000	0.000000
B33	0.000000	0.000000
B43	0.000000	0.000000
B53	1.000000	0.000000

B04	0.000000	0.000000
B14	0.000000	0.000000
B24	0.000000	0.000000
B34	0.000000	0.000000
B44	1.000000	0.000000
B54	0.000000	0.000000
B05	0.000000	0.000000
B15	0.000000	0.000000
B25	0.000000	0.000000
B35	0.000000	0.000000
B45	0.000000	0.000000
B55	1.000000	0.000000
B06	1.000000	0.000000
B16	0.000000	0.000000
B26	0.000000	0.000000
B36	0.000000	0.000000
B46	0.000000	0.000000
B56	0.000000	0.000000

ROW	SLACK OR SURPLUS	DUAL PRICES
2)	3.000000	0.000000
3)	0.000000	0.000000

4)	0.000000	0.000000
5)	0.000000	0.000000
6)	0.000000	0.000000
7)	0.000000	0.000000
8)	0.000000	0.000000
9)	0.000000	0.000000
10)	0.000000	0.000000
11)	0.000000	0.000000
12)	0.000000	0.000000
13)	0.000000	0.000000
14)	0.000000	0.000000

NO. ITERATIONS= 4

BRANCHES= 0 DETERM.= 1.000E 0

b) Five items S = { 20, 10, 15, 10, 5} and bin capacity of 20

interpretation of results: The minimum number of bins used was 3 bins. In one bin, there is the first item. In another bin, there is the second and fourth items. In a third bin, there is the third and the fifth items.

input

MIN a1 + a2 + a3 + a4 + a5

ST

a1 + a2 + a3 + a4 + a5 > 0

```
20b01 + 10b02 + 15b03 + 10b04 + 5b05 - 20a1 <= 0

20b11 + 10b12 + 15b13 + 10b14 + 5b15 - 20a2 <= 0

20b21 + 10b22 + 15b23 + 10b24 + 5b25 - 20a3 <= 0

20b31 + 10b32 + 15b33 + 10b34 + 5b35 - 20a4 <= 0

20b41 + 10b42 + 15b43 + 10b44 + 5b45 - 20a5 <= 0

b01 + b11 + b21 + b31 + b41 = 1

b02 + b12 + b22 + b32 + b42 = 1

b03 + b13 + b23 + b33 + b43 = 1

b04 + b14 + b24 + b34 + b44 = 1

b05 + b15 + b25 + b35 + b45 = 1
END

INT a1
INT a2
```

INT a3

INT a4

INT a5

INT b01

INT b11

INT b21

INT b31

INT b41

INT b02

INT b12

INT b22

INT b32

INT b42

INT b03

INT b13

INT b23

INT b33

INT b43

INT b04

INT b14

INT b24

INT b34

INT b44

INT b05

INT b15

```
INT b25

INT b35

INT b45
output:

 LP OPTIMUM FOUND AT STEP     20
 OBJECTIVE VALUE =   3.00000000


 NEW INTEGER SOLUTION OF    3.00000000     AT BRANCH      0 PIVOT      20
 RE-INSTALLING BEST SOLUTION...

        OBJECTIVE FUNCTION VALUE

        1)      3.000000

  VARIABLE        VALUE          REDUCED COST
        A1         1.000000          1.000000
        A2         1.000000          1.000000
        A3         1.000000          1.000000
```

A4	0.000000	1.000000
A5	0.000000	1.000000
B01	0.000000	0.000000
B11	1.000000	0.000000
B21	0.000000	0.000000
B31	0.000000	0.000000
B41	0.000000	0.000000
B02	1.000000	0.000000
B12	0.000000	0.000000
B22	0.000000	0.000000
B32	0.000000	0.000000
B42	0.000000	0.000000
B03	0.000000	0.000000
B13	0.000000	0.000000
B23	1.000000	0.000000
B33	0.000000	0.000000
B43	0.000000	0.000000
B04	1.000000	0.000000
B14	0.000000	0.000000
B24	0.000000	0.000000
B34	0.000000	0.000000
B44	0.000000	0.000000
B05	0.000000	0.000000

B15	0.000000	0.000000
B25	1.000000	0.000000
B35	0.000000	0.000000
B45	0.000000	0.000000

ROW	SLACK OR SURPLUS	DUAL PRICES
2)	3.000000	0.000000
3)	0.000000	0.000000
4)	0.000000	0.000000
5)	0.000000	0.000000
6)	0.000000	0.000000
7)	0.000000	0.000000
8)	0.000000	0.000000
9)	0.000000	0.000000
10)	0.000000	0.000000
11)	0.000000	0.000000
12)	0.000000	0.000000

NO. ITERATIONS= 20

BRANCHES= 0 DETERM.= 1.000E 0