CS 325 Spring 2019, Homework 8

June 1, 2019

In the bin packing problem, items of different weights (or sizes) must be packed into a finite number of bins, each with the capacity C in a way that minimizes the number of bins used. The decision version of the bin packing problem (deciding if objects will fit into <= k bins) is NP-complete. There is no known polynomial time algorithm to solve the optimization version of the bin packing problem. In this homework you will be examining three greedy approximation algorithms to solve the bin packing problem.

- **First-Fit:** 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.
- **First-Fit-Decreasing:** First sort the items in decreasing order by size, then use First-Fit on the resulting list.
- **Best-Fit:** Place the items in the order in which they arrive. Place the next item into the bin which will leave the least room left over after the item is placed in the bin. If it does not fit in any bin, start a new bin.

1. Give pseudo code and the running time for each of the approximation algorithms.

   - **First-Fit**

```
FIRST-FIT(W[1..n], C)
    r = 0
    let s[1..n] be a new array
    s[1..n] = C
    for i = 1 to n:
        for j = 1 to n:
            if s[j] >= W[i]:
                s[j] = s[j] - W[i]
                if j > r:
                    r = j
                break
    return r
```

   The running time of First Fit is $O(n^2)$, but First Fit can be implemented in $O(n \log n)$ time using Self-Balancing Binary Search Trees.

- **First-Fit-Decreasing**

```
FIRST-FIT-DECREASING(W[1..n], C)
    sort W in descending order
    r = 0
    let s[1..n] be a new array
    s[1..n] = C
    for i = 1 to n:
        for j = 1 to n:
            if s[j] >= W[i]:
                s[j] = s[j] - W[i]
                if j > r:
                    r = j
                break
    return r
```

The running time of First Fit Decreasing is $O(n^2)$, but First Fit Decreasing can be implemented in $O(n \log n)$ time using Self-Balancing Binary Search Trees.

- **Best-Fit**

```
BEST-FIT(W[1..n], C)
    r = 0
    let s[1..n] be a new array
    s[1..n] = C
    for i = 1 to n:
        min = C+1
        bi = 0
        for j = 1 to r:
            if s[j] >= W[i] and s[j] - W[i] < min:
                bi = j
                min = s[j] - W[i]
        if min = C+1:
            r = r + 1
            s[r] = s[r] - W[i]
        else:
            s[bi] = s[bi] - W[i]
    return r
```

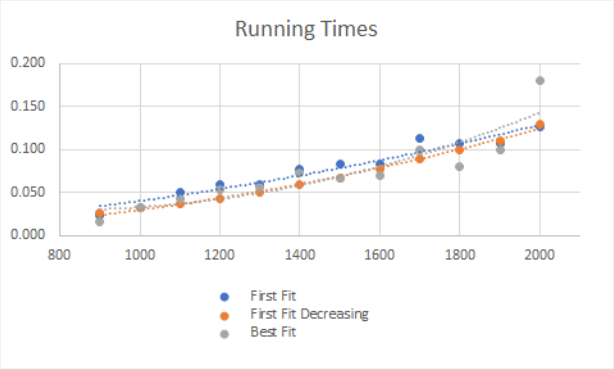
The running time of Best Fit is $O(n^2)$, but Best Fit can also be implemented in $O(n \log n)$ time using Self-Balancing Binary Search Trees.

2. Implement the bin packing approximation algorithms in Python, C++ or C. Your program named binpack should read in a text file named bin.txt with multiple test cases as explained below and output to the terminal the number of bins each algorithm calculated for each test case. Your code should also collect and output the running time of each algorithm. Submit a README file and your program to TEACH.

3. Randomly generate at least 20 bin packing instances of varying sizes (number of items). Submit a description of how the inputs not the code used to produce the random inputs.

   First the capacity of the bin is generated by selecting a random number from 100 to 200. Then for each instance, the number of items is going to increase from 100 to 2000 with an increment of 100. The weights of the items are generated by selecting a random number from 1 to capacity.

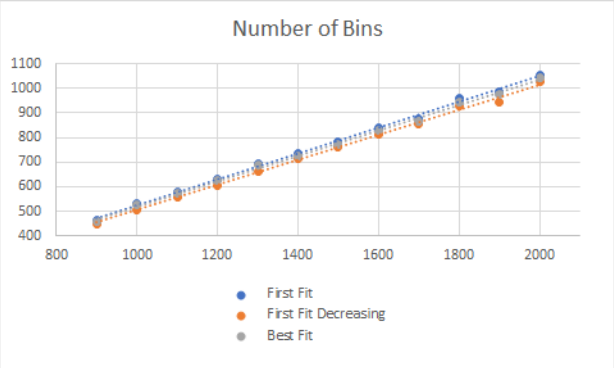
   - Create a graph or chart that shows the number of bins used by each algorithm with instances of different sizes. Which algorithm performs better if the metric is number of bins?



Running Times.

   If the metric is number of bins, FIRST-FIT-DECREASING performs better than BEST-FIT followed by FIRST-FIT.

- Create a graph or chart that shows the running time of each algorithm as a function of number of items. Which algorithm performs better if the metric is running time?



No. of Bins.

If the metric is running time, FIRST-FIT performs better than FIRST-FIT-DECREASING followed by BEST-FIT.

An exact solution to the bin packing optimization problem can be found using 0-1 integer programming. Write an integer program for each of the following instances of bin packing and solve with the software of your choice. Submit a copy of the code and interpret the results.

1. Six items S = {4, 4, 4, 6, 6, 6} and bin capacity of 10

   $$

$$\text{minimize } B = \sum_{i=1}^{n} y_i \tag{1}$$

$$\text{subject to } B \geq 1, \tag{2}$$

$$\sum_{j=1}^{n} a_j x_{ij} \leq V y_i, \forall i \in \{1, ..., n\} \tag{3}$$

$$\sum_{i=1}^{n} x_{ij} = 1, \forall j \in \{1, ..., n\} \tag{4}$$

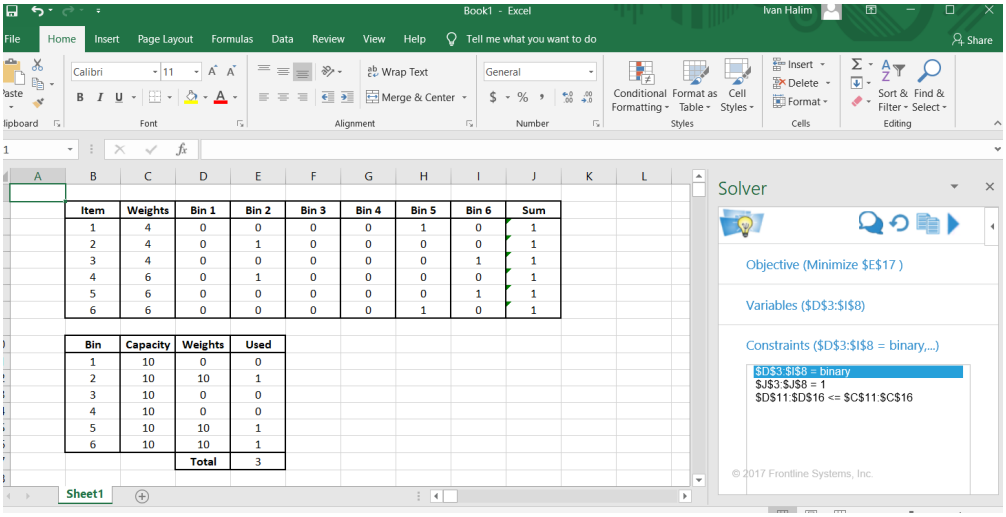
$$y_i \in \{0, 1\}, \forall i \in \{1, ..., n\} \tag{5}$$

$$x_{ij} \in \{0, 1\}, \forall i \in \{1, ..., n\} \forall j \in \{1, ..., n\} \tag{6}$$

$$\tag{7}$$

   $$

   where $y_i = 1$ if bin i is used and $x_{ij} = 1$ if item j is put into bin i.

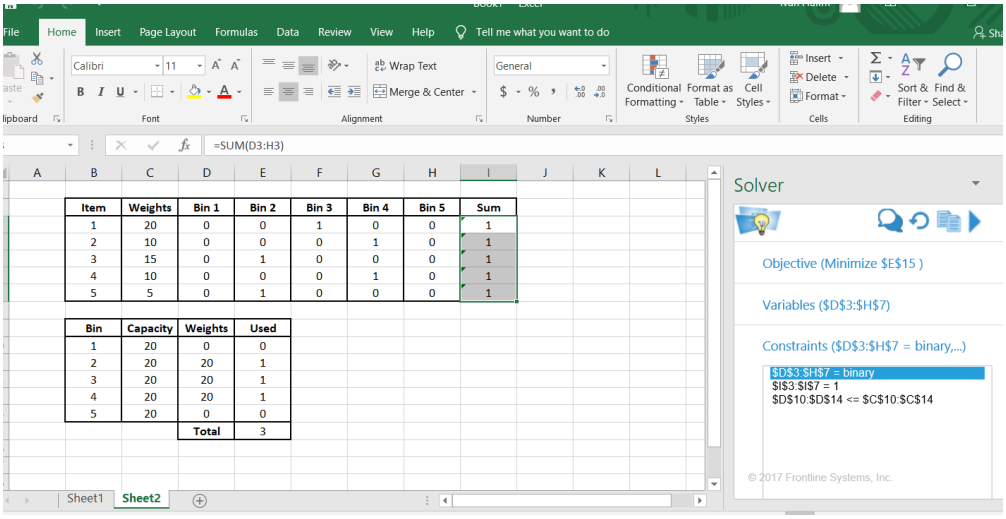


Item	Weights	Bin 1	Bin 2	Bin 3	Bin 4	Bin 5	Bin 6	Sum
1	4	0	0	0	0	1	0	1
2	4	0	1	0	0	0	0	1
3	4	0	0	0	0	0	1	1
4	6	0	1	0	0	0	0	1
5	6	0	0	0	0	0	1	1
6	6	0	0	0	0	1	0	1

Bin	Capacity	Weights	Used
1	10	0	0
2	10	10	1
3	10	0	0
4	10	0	0
5	10	10	1
6	10	10	1
		Total	3

Bin Packing 1.

2. Five items S = {20, 10, 15, 10, 5} and bin capacity of 20



Item	Weights	Bin 1	Bin 2	Bin 3	Bin 4	Bin 5	Sum
1	20	0	0	1	0	0	1
2	10	0	0	0	1	0	1
3	15	0	1	0	0	0	1
4	10	0	0	0	1	0	1
5	5	0	1	0	0	0	1

Bin	Capacity	Weights	Used
1	20	0	0
2	20	20	1
3	20	20	1
4	20	20	1
5	20	0	0
		Total	3

Bin Packing 2.