Arthur Liou
CS325

HW 8

Problem 1: *(20 pts)* 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 two 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.

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

First-Fit

- Create a new bin
- While list is not empty
    - Please first item received in current bin.
    - If not possible, place in the next bin || place in a new bin
- Return list of bins

First-Fit-Decreasing

- Sort items by decreasing order size
- Call first-fit on the resulting list (or just reimplement first-fit)

Best-Fit

- Create a new bin
- Create list of bins & an array for their respective capacities left
- While list is not empty
    - Iterate through Capacity_Left array & insert into the bin which will leave the last left over
    - If does not fit into a bin, create a new bin
- Return list of bins

b) Implement the 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. Submit a README file and your program to TEACH.

Note: Create 20 test cases to fulfill 1C.

See binpack.py

Note: In my code, I left 1b and 1c commented in, so you will see both results for 1b and 1c. If you would like to see just one or the other, please comment out the process() command on line 133 or the block of 20 test cases.

c) Randomly generate at least 20 bin packing instances. Summarize the results for each algorithm. Which algorithm performs better? How often? *Note: Submit a description of how the inputs were generated not the code used to produce the random inputs.*
First Fit: Typically the first few bins have the most items
First Fit Descending: More even distribution of items
Best Fit: Also typically the first few bins have more items, but not necessarily,
Judging from the results, most algorithms performed approximately the same, but first fit descending and best fit typically performed better. First fit descending performed a little better in certain test cases
Description: The inputs and capacity were randomly generated manually, with the amount of inputs ranging from 5-10, and capacity well higher than the largest number in the range.

Note: In my code, I left 1b and 1c commented in, so you will see both results for 1b and 1c. If you would like to see just one or the other, please comment out the process() command on line 133 or the block of 20 test cases.

Problem 2: *(10 pts)* An exact solution to the bin packing optimization problem can be found using 0-1 integer programming (IP) see the format on the Wikipedia page.

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.
Code attached as bin2.py
a) Six items S = { 4, 4, 4, 6, 6, 6} and bin capacity of 10
Sum 30. Requires at least **3 bins**, each bin being Bin(sum=10, items=[6, 4])

b) Five items S = { 20, 10, 15, 10, 5} and bin capacity of 20
Sum 60. Requires at least **3 bins**:
Bin(sum=20, items=[20])
Bin(sum=20, items=[15, 5])
Bin(sum=20, items=[10, 10])

Note: The version of LINDO that you have access to on the OSU server has a limit of 50 integer variables. Therefore, LINDO will only be able to solve problems with at most 6 items.