

Computer Algorithms in Systems Engineering

Spring 2010

Problem Set 7: Branch and bound: Staging building materials

Due: 12 noon, Friday, May 8, 2010

Problem statement

This homework explores the sum of subsets problem, which arises in many contexts. We will use a truck loading problem. A large number of trucks is used to carry structural elements and many types of equipment for construction projects. In many projects, this is a significant portion of the total cost.

You must ship a series of items from a staging area to a construction site, each with a given weight, in the minimum number of trucks. Each truck has the same capacity. We assume that weight is the limiting factor, not the size of the shipment. For example, a truck may hold a maximum of 48,000 pounds of freight. We omit the thousands in the examples that follow, so we consider the truck to have a capacity of 48. If we have items that weigh 14, 12, 11, 6, 3 and 2, they can fit exactly on the truck.

The input to the problem is 1,000 integers, which give the weight of each of 1,000 items to be shipped to the construction site. You are to design an algorithm to place these items on trucks with the maximum possible efficiency (i.e., using the smallest number of trucks). Suggestions:

1. Use a greedy algorithm to place 'obvious' combinations of items on a truck. For example, if a truck has a capacity of 48, you can assign it 48 weight-1 items or 24 weight-2 items, etc. Or you can assign it 12 weight-2 items and 6 weight-4 items.
2. After your greedy algorithm has handled the obvious combinations, you will be left with a smaller number of integers. Implement the 'sum of subsets' algorithm described below using a branch and bound framework to assign this next set of items to trucks, again with maximum efficiency.
3. Last, there will be some items left over that don't fit on a truck exactly. Find a way to assign them to trucks so that there is minimal inefficiency.

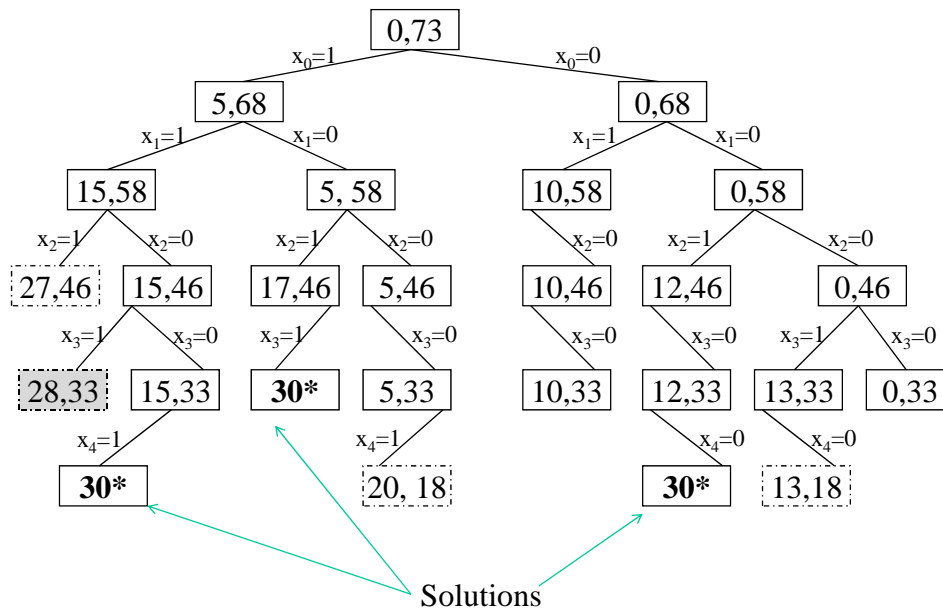
We don't provide any hints for the greedy algorithm; this homework allows you to exercise some of the algorithm design knowledge you've been acquiring this semester.

Sum of subsets

The sum of subsets algorithm is described below, and in section 7.3 of the Horowitz text. It uses the backtracking version of branch and bound. The key steps are:

- Sort the weights in ascending order. There will be ties, since many items may have the same weight.

- In the branch and bound tree, whether to put each integer in the solution is a decision represented by a node and arcs to its two children, representing $x_i=1$ and $x_i=0$. The root node is a sentinel with weight 0. All real items have weights > 0 .
- The figure below shows an example branch and bound tree.
 - It assumes a set of items with 6 weights: $\{5, 10, 12, 13, 15, 18\}$. They have been sorted in ascending order. Each item has a different weight, to make the example clearer, but this is not required in the problem.
 - The truck capacity is 30.
 - In each node, we show the weight of the items already in the solution and the weight of all the items still to be considered after this. Thus, in the root node, there is 0 weight in the solution and 73 units of weight left to be considered.
 - Use an array or ArrayList x to keep track of the solution at each point in the tree. When $x_i=1$, the item is in the solution; if $x_i=0$, the item is not in the solution.



We arbitrarily put the first weight, 5, in the solution. We do this by setting $x_0=1$. At this node, the left child of the root, we now have 5 units in the solution, and 68 more to consider. We next put weight 10 in the solution, generating the next left child. At this node (15,58), there are 15 units in the solution and 58 to consider. We next try to put weight 12 in the solution to generate node (27, 46). However, we can't go farther down this branch of the tree because we know the next weight, 13, is greater than 12 (because the weights are sorted in ascending order) and will

certainly make the sum of that solution, 27, plus 13 exceed the target, 30. We thus don't generate any left children from (15,58).

We next generate the right child of node (15,58) by excluding weight 12 from the solution, to obtain node (15,46). We then try to add weight 13 to get node (28,33), but we can't go any farther, because the next weight, 15, which is larger than 13, would make the sum exceed 30. We thus don't generate any left children from (15,46), and instead we exclude weight 13. This generates a right child (15,33), and by including the next weight, 15, we reach a solution. This solution has $x_0=1$, $x_1=1$, $x_2=0$, $x_3=0$ and $x_4=1$; it includes weights 5, 10 and 15.

You can go through the rest of the example to see how the algorithm goes forward generating left children until there are no further solutions, and then it backtracks, generates a right child, and continues. The algorithm eventually traverses the entire tree that it has generated and terminates. Note that the full decision tree, which in this example would be of depth 6 along all paths, is not generated, because we can terminate the generation of nodes early when we are sure that no further solutions exist along a path by using a set of pruning rules.

At a node, you can apply three pruning rules to determine whether you should generate any children. If we let:

- s= sum of the weights already in the solution, (first quantity shown in the nodes)
- r= sum of the weights remaining to be considered, (second quantity)
- W_k = weight of node k (k denotes current node)
- M= target sum of weights (truck capacity)

Then the three pruning rules may be stated as:

1. If $(s + W_k = M)$, output solution
2. If $(s + W_k + W_{k+1} < M)$, generate left child ($x_k = 1$)
3. If $(s + r - W_k \geq M)$ and $(s + W_{k+1} \leq M)$, generate right child ($x_k = 0$)

Rule 1 states that if the solution so far, plus the current weight, equals the target, output it.

Rule 2 states that if the solution so far, plus the current weight, plus the next weight, are less than the target, generate the left child and continue.

Rule 3 has two parts.

Part a: If the solution so far plus the remaining weights without the current weight is greater than or equal to the target, and

Part b: If the solution so far plus the next weight does not exceed the target, generate the right child and continue. No solution can be found by going to the left child.

Rules 1 and 2 are mutually exclusive. Rule 3 applies at all nodes, regardless of rules 1 and 2. If none of the rules are satisfied at a node, then you've reached the end of a path. Backtrack to the parent node and try the rules there that you haven't already tried. (Normally you checked rules 1 and 2 when you first created the node, so you would just need to check rule 3.) Once you've backtracked to the root, you're done. These pruning rules depend on the weights being sorted in ascending order.

Assignment

Design an algorithm to solve this problem and implement it in a Java program. The steps below are suggested steps, but you are free to use another approach.

1. The item weights are in a file itemweight.txt on the course website. The weights are sorted in increasing order.
2. Design and write a greedy algorithm to put many of the 1000 items onto full trucks using simple rules.
3. Then design and write a branch and bound algorithm to handle the more difficult combinations of items that remain to be assigned to trucks. Use the sum of subsets algorithm repeatedly to place as many items as possible on trucks so that truck capacity is used fully.
4. Last, design and write an algorithm to handle any remaining items; it can be greedy, branch and bound, or a simple heuristic.
5. Output the number of trucks used, how many have unused capacity, and the percent of truck capacity that is used. Output the weights assigned to each truck to a text file. Check that the sum of the items assigned to each truck is equal to the sum of the weights in the input, as a correctness check .
6. Test your solution with truck capacities between 48 and 60 using the set of integer item weights provided on the course Web site.

We provide the performance of our solution for different truck capacities between 48 and 60 below. You should be able to come close to 100% efficiency in your solution.

Truck capacity	Fully loaded trucks	Partially loaded trucks	Total trucks	Percent truck capacity used
48	315	1	316	99.80
49	308	1	309	99.97
50	302	1	303	99.91
51	296	1	297	99.93
52	291	1	292	99.69
53	285	1	286	99.86
54	280	1	281	99.76
55	275	1	276	99.72
56	270	1	271	99.74
57	265	1	266	99.83
58	260	1	261	99.99
59	256	1	257	99.83
60	247	6	253	99.72

Turn In

1. Please include a file briefly explaining your algorithm and its implementation.
2. Place a comment with your full name, Stellar username, and assignment number at the beginning of all .java files in your solution.
3. Place all of the files in your solution in a single zip file.
 - a. Do not turn in electronic or printed copies of compiled byte code (.class files) or backup source code (.java~ files)
 - b. Do not turn in printed copies of your solution.
4. Submit this single zip file on the 1.204 Web site under the appropriate problem set number. For directions see **How To: Submit Homework** on the 1.204 Web site.
5. Your solution is due at noon. Your uploaded files should have a timestamp of no later than noon on the due date.
6. After you submit your solution, please recheck that you submitted your .java file. If you submitted your .class file, you will receive **zero credit**.

Penalties

- 30 points off if you turn in your problem set after Friday (May 8) noon but before noon on Monday (May 11). You have two no-penalty two-day (48 hrs) late submissions per term.
- No credit if you turn in your problem set after noon on Monday.

MIT OpenCourseWare
<http://ocw.mit.edu>

1.204 Computer Algorithms in Systems Engineering
Spring 2010

For information about citing these materials or our Terms of Use, visit: <http://ocw.mit.edu/terms>.