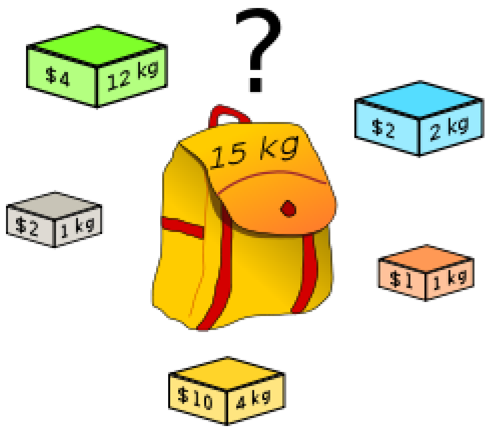
CS21003
Tutorial 2
August 4th, 2017

1. There are n jobs and one processor. Assume that all jobs are available at the start time, and time required by each job is also known. For example, there may be 4 jobs with t_1= 5, t_2= 3, t_3= 7, t_4= 2. The goal is to schedule ***all these jobs*** on the processor so as to minimize the average waiting time of jobs in the processors. Propose a greedy solution to the problem and argue for its optimality. What will be the waiting time for the example in the problem?

2. Consider another variant of Problem 1 above, where each of these n jobs also comes with a deadline d_i (along with time t_i required to process it), by which the job should be completed. A job is late if its finish time is after the deadline, and the lateness is defined by the difference between its finish time and the deadline. Suppose your objective is to minimize the maximum lateness for any of the jobs. E.g., if (d_1=5, t_1=4) and (d_2=6, t_2=4), it is easy to see that you cannot avoid lateness. If you schedule the first job 1st, lateness for the 2nd job is 2, while if you schedule the second job first, lateness for first job is 3. So, the minimum value of maximum lateness among all jobs is 2. Now, suppose these greedy strategies are known to you:
i). Schedule the job with the minimum processing time first.
ii). Schedule the job with the minimum slack time (d_i-t_i) first.
iii). Schedule the job with the earliest deadline first.

Which of these greedy strategies would give you an optimal solution and which will not? To prove your argument, either find a counter-example or prove optimality.

3. Consider a knapsack problem. You are given a bag with a predefined maximum capacity. For example, in this figure, the bag has a maximum capacity of 15 Kg. You are given various items with their weights as well as the values. Your objective is to fill the bag using these items such that the overall value of the items is maximized, constrained by the maximum capacity of the bag. This problem can have many variants, we consider the following:



a. We have N objects and a knapsack. The i^{th} object has positive weight w^i and positive unit value v^i. The knapsack maximum capacity is C. We wish to select a set of fractions of objects to put in the knapsack so that the total value is maximized without breaking the knapsack. Think of a greedy solution to this problem.

For an example input of 5 objects with their weights and values given below

Weight 10 20 30 40 50

Value 20 30 66 40 60

And a maximum capacity of 100, what is the maximum value that can be packed into the knapsack using your strategy?

b. Show by giving an example with 3 items that your greedy algorithm does not always provide an optimal solution to the Knapsack problem if fractions of objects are not allowed to put in the knapsack. This is called **0-1 Knapsack problem.**

4. You are given a sequence of n songs where the ith song is L_i minutes long. You want to place all of the songs on an ordered series of CDs (e.g., CD_1, CD_2, CD_3, ..., CD_k) where each CD can hold m minutes. Furthermore,
(1) The songs must be recorded in the given order, $song_1$, $song_2$, ..., $song_n$.
(2) All songs must be included.
(3) No song may be split across CDs.
Your goal is to determine how to place them on the CDs as to minimize the number of CDs needed. Propose a greedy algorithm to solve this problem. Further, suppose my new objective is to minimize the overall blank spaces in the CDs (except the last CD). Will the same strategy still work? Is it also an optimal algorithm?

5. You given as input n real numbers $x_1, \ldots, x_n$. Design an efficient algorithm that uses the minimum number m of unit intervals $[L_i, L_{i+1})$ $(1 \le i \le m)$ that cover all the input numbers. A number x_j is covered by an interval $[L_i, L_{i+1})$ if $L_i \le x_j < L_{i+1}$.

6. Consider the recurrence relation T(0) = T(1) = 2 and for n > 1

$$T(n) = \sum_{i=1}^{n-1} T(i)*T(i-1)$$

We consider the problem of computing T(n) from n.
(a) Show that if you implement this recursion directly in say the C programming language, that the program would use exponentially, in n, many arithmetic operations.
(b) Give an algorithm for this problem that only uses O(n) arithmetic operations