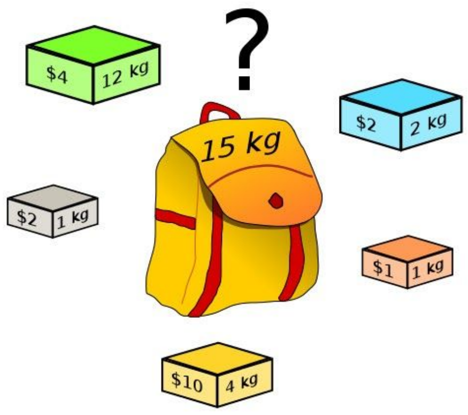
Topic 24: NP-Completeness Solutions

Copyright © 2016, 2017, 2020 Nodari Sitchinava and Daniel D Suthers. These solution notes may only be used by students, TAs and Faculty in ICS 311 Fall 2020 at the University of Hawaii. Please notify suthers@hawaii.edu of any violations of this policy, or of any errors found.

0-1 Knapsack Problem (optimization version): There are *n* items. Item *i* is worth v_i dollars and weighs w_i pounds. Which items should a thief take to maximize the total value of stolen items without exceeding *W* pounds in weight (maximum capacity of his knapsack). It is called 0-1 because the thief must either take or not take each item (they are discrete objects, like gold ingots).



In the lecture notes on Greedy Algorithms we mentioned that picking items greedily by picking the items with maximum value per pound does not give us optimal solution to the 0-1 Knapsack problem. Today we will prove that this problem is NP-Complete, thus, an efficient algorithm is very unlikely. First, we need a decision version of the problem:

0-1 Knapsack Problem (decision version): Given a set of *n* items *S*, each item *i* defined by a value/weight pair (v_i , w_i), and two positive integers *V* and *W,* is there a subset $S' \subseteq S$ of items, such that $\sum_{i \in S'} v_i \geq V$ and $\sum_{i \in S'} w_i \leq W$?

1. (2.5pts) Prove that 0-1 Knapsack problem is in NP, by describing a polynomial time algorithm to verify the "language"

The verification algorithm A(x,y) takes two arguments:

- x = description of an instance of the problem (a set of *n* pairs (v_i , w_i), and integers *V* and *W*). Assume we access these with object notation: x.pairs, x.V and x.W.
- *y* = a "certificate" for a solution.

a. (0.5) What would the certificate contain?

Answer: The certificate will contain the subset of pairs (v_i, w_i) that comprise the proposed answer to the knapsack problem, or alternatively the indices *i* of such pairs in x.

b. (1.5) How would the certificate be checked? Provide pseudocode for A.

Answer: The verification algorithm will go through the pairs (v_i, w_i) of the certificate, ensuring that they are in the problem statement, and summing the v_i and w_i and checking if the sum of v_i is at least V and the sum of w_i is at most W. For example:

```
A(x,y)
V-sum = 0
W-sum = 0
for each ($v_i$, $w_i$) $\in$ y
    if ($v_i$, $w_i$) $\notin$ x.pairs return false
    V-sum = V-sum + $w_i$
    W-sum = W-sum + $w_i$
if (V-sum >= x.V) and (W-sum <= x.W)
    return true
else return false
```

c. (0.5) Prove that your pseudocode is polynomial time.

The algorithm requires a single O(n) scan through the certificate, with constant time operations except set membership in x.pairs, which at most is O(n). So it is at worst $O(n^2)$.

2. (2.5 pts) To **prove that 0-1 Knapsack is NP-hard** we will reduce the Subset Sum problem to the Knapsack problem. The input to Subset Sum problem is a sequence of *n* integers a_1 , a_2 , …, a_n and an additional *t*. The decision version of Subset Sum answers "yes" if there is a subset of a_1 , a_2 , …, a_n that sum to *t*.

Given the input to Subset Sum, our reduction algorithm will create an instance of the 0-1 Knapsack problem by setting $V = W = t$ and creating pairs (a_1, a_1), (a_2, a_2), …, (a_n, a_n).

a. (0.5) How long does this reduction algorithm take?

Answer: It takes linear time: a single scan through the input to convert integers a_i into pairs (a_i, a_i) and constant time to copy t to V and W.

Now we just need to prove that there is a 'yes' answer to the 0-1 Knapsack Problem on our new input if and only if there is a 'yes' answer to the Subset Sum problem on input $(a_1, a_2, \ldots, a_n, t)$:

b. (1.0) Prove that an existence of a 'yes' answer to 0-1 Knapsack Problem on input $((a_1, a_1), (a_2, a_2), \ldots, (a_n, a_n), V = t, W = t)$ implies an existence of a 'yes' answer to the Subset Sum problem on input $(a_1, a_2, \ldots, a_n, t)$.

Answer: A 'yes' answer to the 0-1 Knapsack Problem on input $((a_1, a_1), (a_2, a_2), \ldots, (a_n, a_n), V = t, W = t)$ means that we can pick a subset of pairs S', such that $\sum_{(a_i, a_i) \in S'} a_i \geq V = t$ and $\sum_{(a_i, a_i) \in S'} a_i \leq W = t$. I.e. $\sum_{a_i \in S'} a_i = t$ (if both ≥ and ≤ are true, the values must be equal). That means picking the same indices *i* for the Subset Sum problem will also be a correct solution, and therefore, there is a 'yes' answer to the Subset Sum problem on input $(a_1, a_2, \ldots, a_n, t)$

c. (1.0) Prove that an existence of a 'yes' answer to Subset Sum problem on input $(a_1, a_2, \ldots, a_n, t)$ implies an existence of a 'yes' answer to 0-1 Knapsack Problem on input $((a_1, a_1), (a_2, a_2), \ldots, (a_n, a_n), V = t, W = t)$

Answer: A 'yes' answer to the Subset Sum problem on input $(a_1, a_2, \ldots, a_n, t)$ means there is a subset S' among $\{a_1, a_2, \ldots, a_n\}$, such that $\sum_{a_i \in S'} a_i = t$. But that means that $\sum_{a_i \in S'} a_i \geq V$ and $\sum_{a_i \in S'} a_i \leq W$, because $V = t$ and $W = t$. Then picking the same indices in the 0-1 Knapsack Problem on input $((a_1, a_1), (a_2, a_2), \ldots, (a_n, a_n), t, t)$ will result in a valid solution, i.e. a 'yes' answer to the 0-1 Knapsack Problem exists.

Optional: If you finished early, consider the following problem:

3. Challenge Problem: Consider this decision problem:

Partition Problem: Given a multiset *S* of *n* positive integers $\{a_1, a_2, \ldots, a_n\}$ does there exist a submultiset *S'*, such that the sum of items in *S'* equals to the sum of items not in *S'*. (A multiset is a set that allows duplicate items.)

a. (1.0) What is wrong with the following reduction from Subset Sum to prove that Partition problem is NP-hard:

Given a multiset *S* of *n* positive integers {$a_1, a_2, \ldots, a_n$}, the reduction algorithm sets $K = 0.5 * \sum_{i=1}^{n} a_i$ and defines the input to Subset Sum as ({$a_1, a_2, \ldots, a_n$}, K/2).

Answer: The reduction goes the wrong way: we are reducing the Partition problem to the Subset Sum problem. I.e. with this reduction we can show that Partition $\leq_p$ Subset-Sum. I.e. Partition is no harder than Subset Sum. It does not prove that Partition is hard to solve.

b. (1.0) Here is another wrong proof. What is wrong with it?

Given the input to Subset Sum problem ({$a_1, a_2, \ldots, a_n$}, t), consider the subset *S'* of items a_i, such that $\sum_{a_i \in S'} a_i = t$. The reduction algorithm creates multiset $S = S' \cup S'$; i.e. it duplicates each item in *S'*. An answer 'yes' to this instance of Partition exists if and only if there is a 'yes' answer to Subset Sum on input *({$a_1, a_2, \ldots, a_n$}, t)*.

Answer: It is a valid reduction, however, to construct the multiset *S* we must solve the Subset Sum problem ("consider the subset *S'* of items a_i, such that $\sum_{a_i \in S'} a_i = t$"), for which no polynomial time solution is known to exist. Therefore, we don't know how to perform the reduction in polynomial time. For a valid NP-hardness proof, the reduction should be oblivious to the actual solution and still work.