Knapsack Problem

Dr. Bibhudatta Sahoo

Communication & Computing Group

Department of CSE, NIT Rourkela

Email: bdsahu@nitrkl.ac.in, 9937324437, 2462358

Greedy Algorithm

- A greedy algorithm for an optimization problem always makes the choice that looks best at the moment and adds it to the current sub solution.
- A greedy algorithm always makes the choice that looks best at the moment. That is it makes a locally optimal choice that may be lead to a globally optimal solution. This algorithm is simple and more efficient compared to other optimization algorithm
- Greedy algorithms don't always yield optimal solutions but, when they do, they're usually the simplest and most efficient algorithms available.
- This heuristic strategy does not always produce an optimal solution, but sometimes it does. There are two key ingredients in greedy algorithm that will solve a particular optimization problem.

[1] Greedy choice property **[2] Optimal substructure**

Greedy choice property

- A globally optimal solution can be arrived at by making a locally optimal (greedy) choice. In other words, when a choice is to be made, then it looks for best choice in the current problem, without considering results from the sub-problems.
- In this algorithm choice is made that seems best at the moment and solve the sub-problems after the choice is made. The choices made by a greedy algorithm may depend on choices so far, but it can not depend on any future choice or solution to the sub-problems.
- The algorithm progress in a **top down** manner, making one greedy choice one after another, reducing each given problem instances into smaller one.

Optimal substructure:

- A problem is said to have **optimal substructure** if an optimal solution can be constructed efficiently from optimal solution to its sub-problem.
- The optimal substructure varies across problem domain in two ways:
- (i) How many sub-problems are used in an optimal solution to the original problem.
- (ii) How many choices we have in determining which sub-problem to use in an optimal solution.
- In Greedy algorithm a sub-problem is created by having made the greedy choice in the original problem. Here, an optimal solution to the sub-problem, combined with the greedy choice already made, yield an optimal solution to the original problem.

A greedy algorithm has five components:

- A set of candidates, from which to create solutions.
- A selection function, to select the best candidate to add to the solution.
- A feasible function is used to decide if a candidate can be used to build a solution.
- An objective function, fixing the value of a solution or an incomplete solution.
- An evaluation function, indicating when you find a complete solution.

Solution to optimization problem with Greedy

It is one way to construct a feasible solution for such optimization problems, and, *sometimes*, it leads to an optimal one.

When applying this method, we construct a solution in stages. At each stage, we make a decision that appears to be the best *at that time*, according to a certain *greedy criterion*. Such a decision will not be changed in later stages. Hence, each decision should assume the feasibility.

Sometimes, such a *locally optimal* solution does lead to an overall optimal one. ☺

Knapsack problem; introduction

- The knapsack problem is well known in operations research literature. This problem arises whenever there is a resource allocation problem with financial constraints.
- For example, given that you have a fixed budget, how do you select what things you should buy? Assume that everything has a cost and value. We seek assignments that provide the most value for a given budget.
- The term ***knapsack problem*** invokes the image of a backpacker who is constrained by a fixed-size knapsack and so must fill it only with the most useful items.

Knapsack problem; introduction

- The classical knapsack problem assumes that each item must be put entirely in the knapsack or not included at all. It is this 0/1 property that makes the knapsack problem difficult.
- In the **0/1 Knapsack problem**, you are given a bag which can hold W kg. There is an array of items each with a different weight and price value assigned to them.
- The objective of the 0/1 Knapsack is to **maximize the value** you put into the bag. You are allowed to only take all or nothing of a given item.

Knapsack problem

- *Knapsack* problem : Suppose we have n integers $a_1, a_2, \ldots, a_n$ and a constant W. We want to find a subset of integers so that their sum is less than or equal to but is as close to W as possible. There are 2^n subsets. ($n = 100$, $2^n = 1.26765 \times 10^{30}$, it takes 4.01969×10^{12} years if our computer can examine 10^{10} subsets per second.)

- A problem is considered *tractable* (computationally easy, $O(n^k)$) if it can be solved by an efficient algorithm and *intractable* (computationally difficult, the lower bound grows fast than n^k) if there is no efficient algorithm for solving it.

- The class of *NP-complete* problems: There is a class of problems, including TSP and Knapsack, for which no efficient algorithm is currently known.

Knapsack problem

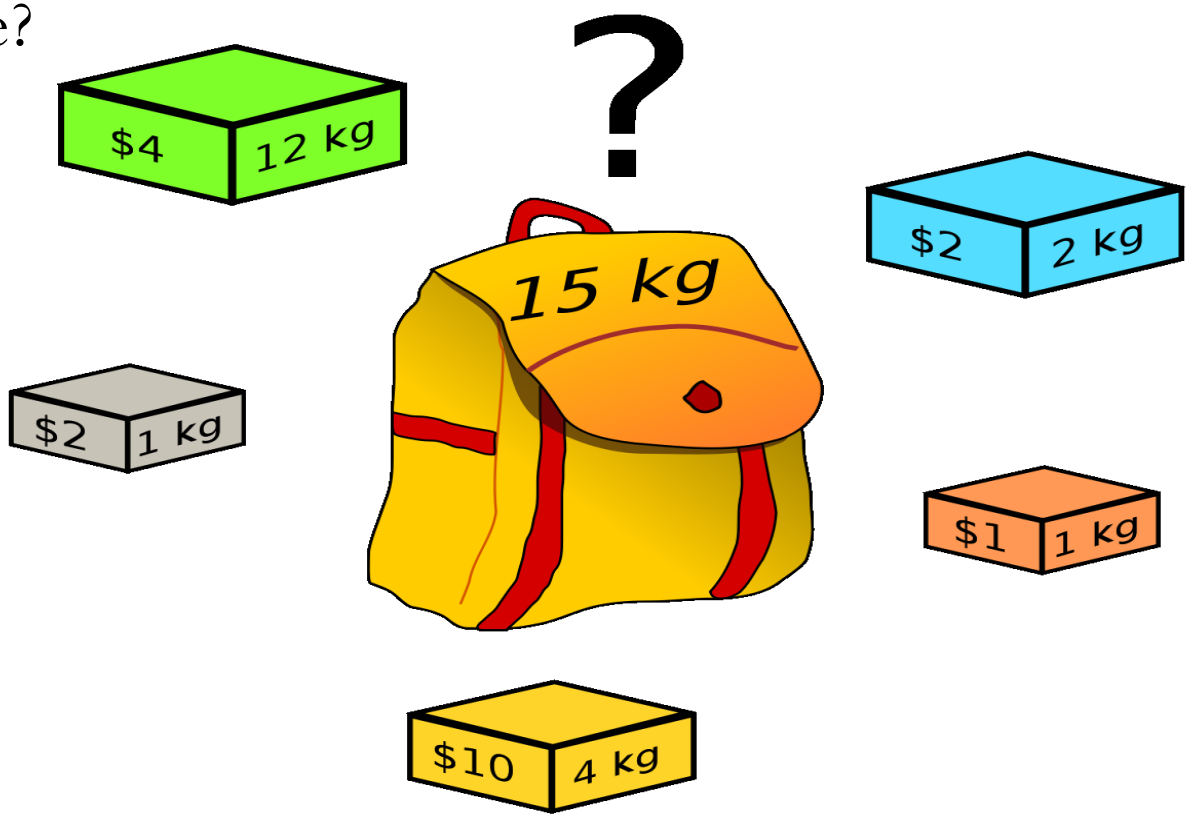
There are two types of knapsack problem.

1. **0-1 knapsack problem**: In 0-1 knapsack problem each item either be taken or left behind.

2. **Fractional knapsack problem**: In fractional knapsack problem fractions of items are allowed to choose.

0-1 knapsack problem

A thief robbing a store finds n items, the i^{th} item is worth v_i dollar and weight w_i pounds, where v_i and w_i are integers. He wants to take as valuable load as possible, but he can carry at most m pounds in his knapsack, where m is an integer. Which item should he take?



0/1 Knapsack problem (

- Given n objects 1 through n, each object i has an integer weight w_i and a real number value p_i, for $1 \leq i \leq n$.
- There is a knapsack with a total integer capacity $\boldsymbol{W}$.
- The 0–1 knapsack problem attempts to fill the sack with these objects within the weight capacity W while **maximizing** the total value of the objects included in the sack, where an object is totally included in the sack or no portion of it is in at all.
- That is, solve the following optimization problem with x_i = **0 or 1**, for $1 \leq i \leq n$:

$$maximize \sum_{i=1}^{n} x_i p_i \quad subjected\ to \sum_{i=1}^{n} x_i w_i \leq W$$

Knapsack problem

- *Knapsack* problem : Suppose we have n integers $a_1, a_2, \ldots, a_n$ and a constant W. We want to find a subset of integers so that their sum is less than or equal to but is as close to W as possible. There are 2^n subsets. ($n = 100$, $2^n = 1.26765 \times 10^{30}$, it takes 4.01969×10^{12} years if our computer can examine 10^{10} subsets per second.)

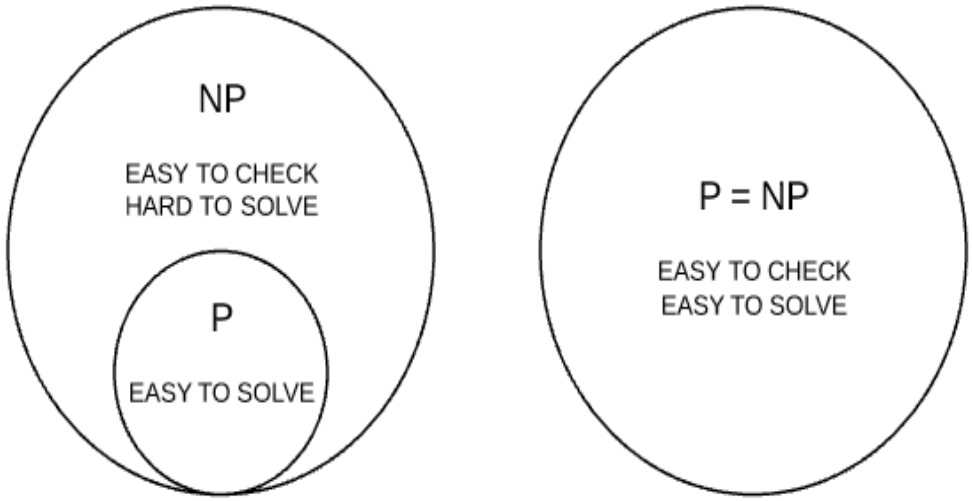
- A problem is considered ***tractable*** (computationally easy, $O(n^k)$) if it can be solved by an efficient algorithm and ***intractable*** (computationally difficult, the lower bound grows fast than n^k) if there is no efficient algorithm for solving it.

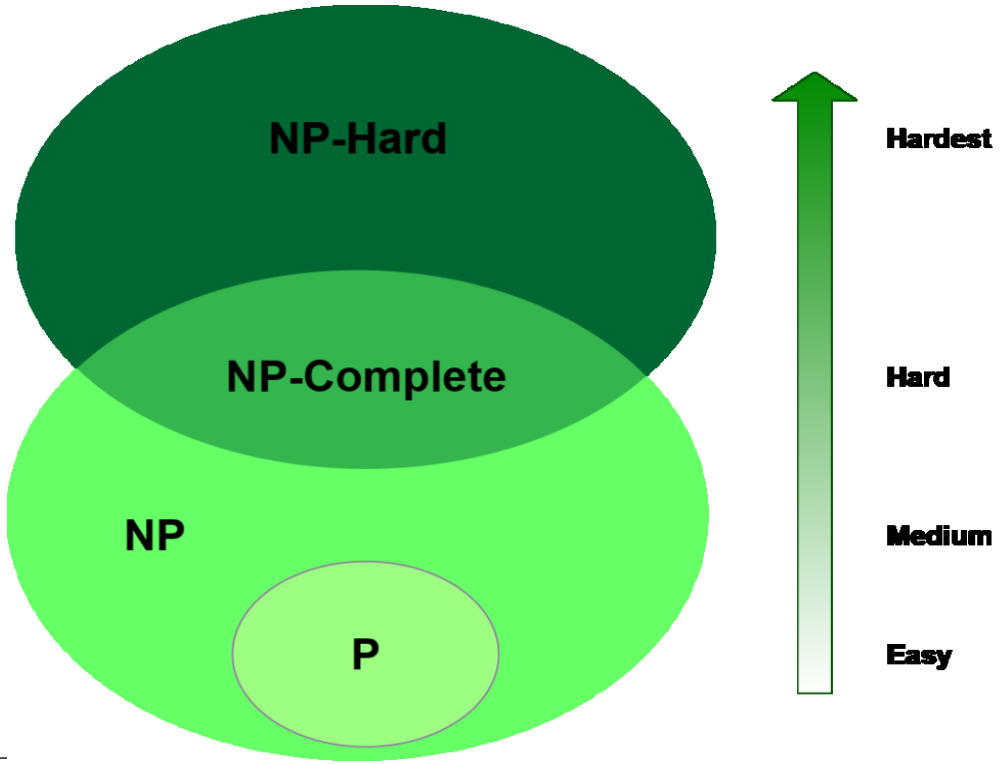
- The class of *NP-complete* problems: There is a class of problems, including TSP and Knapsack, for which no efficient algorithm is currently known.

knapsack problem is NP-complete

- The knapsack problem is NP-complete because the known NP-complete problem **subset-sum** is polynomially reducible to the **knapsack problem**, hence every problem in NP is reducible to the knapsack problem.
- The fact that we know an O(nW) algorithm for knapsack doesn't contradict the possibility that P≠NP because O(nW) is not polynomial in the length of the input to the algorithm, which is O(nlogW). If we ever did find an algorithm that ran in time polynomial in n and logW that would prove that P=NP.
- Algorithms that are polynomial in the magnitudes of coefficients in the input instead of their sizes are called *pseudo polynomial*. An NP-complete problem might or might not have a pseudo polynomial algorithm. Those that don't (assuming P≠NPP≠NP) are *strongly NP-complete*. The knapsack problem is not strongly NP-complete, but it is still NP-complete.

P vs NP Problem





0-1 knapsack problem is pseudo-polynomial

- knapsack problem is NP-complete while it can be solved by DP. They say that the DP solution is pseudo-polynomial, since it is exponential in the "length of input" (i.e. the numbers of bits required to encode the input).
- The running time is O(NW) for an unbounded knapsack problem with N items and knapsack of size W. W is not polynomial in the length of the input though, which is what makes it *pseudo*-polynomial.
- Consider W = 1,000,000,000,000. It only takes 40 bits to represent this number, so input size = 40, but the computational runtime uses the factor 1,000,000,000,000 which is $O(2^{40})$.
- So the runtime is more accurately said to be $O(N.2^{\text{bits in W}})$, which is exponential.

Fractional knapsack problem

$$\text{maximize} \sum_{1 \le i \le n} p_i x_i \tag{4.1}$$

$$\text{subject to} \sum_{1 \le i \le n} w_i x_i \le m \tag{4.2}$$

$$\text{and } 0 \le x_i \le 1, \quad 1 \le i \le n \tag{4.3}$$

The profits and weights are positive numbers.

A feasible solution (or filling) is any set $(x_1, \ldots, x_n)$ satisfying (4.2) and (4.3) above. An optimal solution is a feasible solution for which (4.1) is maximized.

A solution to fractional knapsack problem

Example 4.1 Consider the following instance of the knapsack problem: $n = 3, m = 20, (p_1, p_2, p_3) = (25, 24, 15)$, and $(w_1, w_2, w_3) = (18, 15, 10)$. Four feasible solutions are:

	(x_1, x_2, x_3)	$\sum w_i x_i$	$\sum p_i x_i$
1.	(1/2, 1/3, 1/4)	16.5	24.25
2.	(1, 2/15, 0)	20	28.2
3.	(0, 2/3, 1)	20	31
4.	(0, 1, 1/2)	20	31.5

Of these four feasible solutions, solution 4 yields the maximum profit. As we shall soon see, this solution is optimal for the given problem instance. □

Greedy algorithm in 0-1 knapsack problem

- The greedy algorithm in 0-1 knapsack problem can be applied as follows:

1. **Greedy choice**: Take an item with maximum value per pound.
2. **Optimal substructure**: Consider the most valuable load that weights at most m pounds. These m pounds can be choose from n item. If j^{th} item is choose first then remaining weight $m-w_i$ can be choose from n-1 remaining item excluding j.

Example: Greedy Method with SUBSET Paradigm

- There are $n = 5$ objects with integer weights $w[1..5] = \{1,2,5,6,7\}$, and values $p[1..5] = \{1,6,18,22,28\}$. Assuming a knapsack capacity of 11.

Item/ weight	Status	Profit
1/1	1	1
2/2	1	1+6=7
3/5	1	7+18=25
4/6	0	25
5/7	0	25

GreedyKnapsack(Subset paradigm)

```
1.  Algorithm GreedyKnapsack(m, n)
2.  //p[1 : n] and w[1 : n] contain the profits and weights
3.  // respectively of the n objects
4.  // m is the knapsack size and x[1 : n] is the solution vector.
5.  {
6.  for i := 1 to n do x[i] := 0.0;
7.  U:=m; P:=0;
8.  for i := 1 to n do
9.  {
10. if (w[i] > U) then break;
11. x[i] := 1.0; U.:= U - w[i); P := P + p[i]
12. }
13. Return P;
14. }
```

Example: Greedy Method with ORDERING Paradigm

- There are $n = 5$ objects with integer weights $w[1..5] = \{1,2,5,6,7\}$, and values $p[1..5] = \{1,6,18,22,28\}$. Assuming a knapsack capacity of 11.

Item	w	v	v/w	Status	Profit
5	7	28	4	1	28
4	6	22	3.66	0	
3	5	18	3.6	0	
2	2	6	3	1	28+6=34
1	1	1	1	1	34+1=35

Optimal solution :40

GreedyKnapsack(ordering paradigm)

```
1.  Algorithm GreedyKnapsack(m, n)
2.  //p[1 : n] and w[1 : n] contain the profits and weights
    respectively
3.  //of the n objects ordered such that p[i]/w[i] ≥ p[i +1]/w[i +
    1].
4.  // m is the knapsack size and x[1 : n] is the solution
    vector.
5.  {
6.  for i := 1 to n do x[i] := 0.0;
7.  U:=m; P :=0;
8.  for i := 1 to n do
9.  {
10. if (w[i] > U) then break;
11. x[i] := 1.0; U.:= U - w[i]; P := P + p[i]
12. }
13. Return P;
14. }
```

GreedyKnapsack(ordering paradigm)

```
1   Algorithm GreedyKnapsack(m, n)
2   // p[1 : n] and w[1 : n] contain the profits and weights respectively
3   // of the n objects ordered such that p[i]/w[i] ≥ p[i + 1]/w[i + 1].
4   // m is the knapsack size and x[1 : n] is the solution vector.
5   {
6       for i := 1 to n do x[i] := 0.0; // Initialize x.
7       U := m;
8       for i := 1 to n do
9       {
10          if (w[i] > U) then break;
11          x[i] := 1.0; U := U − w[i];
12      }
13      if (i ≤ n) then x[i] := U/w[i];
14  }
```

m = 15

i	1	2	3
p	15	10	20
w	5	4	10
p/w	3	2.5	2

i	U	w[i]	x[i]
1	15	5	1
2	10	4	1
3	6	10	0.6

P = 15 + 10 + 20 x 0.6 = 37

0-1 knapsack is harder!

- 0-1 knapsack cannot be solved by the greedy strategy
  - Unable to fill the knapsack to capacity, and the empty space lowers the effective value per rupees of the packing
  - We must compare the solution to the sub-problem in which the item is included with the solution to the sub-problem in which the item is excluded before we can make the choice
  - Dynamic Programming

Fractional knapsack problem
or
Continuous knapsack problem

Knapsack Problem: Text Book page

- Suppose there are n objects and a knapsack or bag with maximum capacity m.
- An object i has a weight w_i and p_i is the profit associated with the object i.
- If a fraction x_i, $0 \leq x_i \leq 1$, of object i is place into the knapsack, then a profit of $p_i x_i$ is earned.
- The objective is to obtain a filling of the knapsack that maximizes the total profit earned.
- Since the knapsack capacity is m, we require the total weight of all chosen objects to be at most m.

The problem can be stated as

$$\text{maximize} \sum_{1 \leq i \leq n} p_i x_i \qquad (4.1)$$

$$\text{subject to} \sum_{1 \leq i \leq n} w_i x_i \leq m \qquad (4.2)$$

$$\text{and } 0 \leq x_i \leq 1, \quad 1 \leq i \leq n \qquad (4.3)$$

The fractional knapsack problem

- We have n objects, each with a weight $w_i > 0$; a profit $p_i > 0$; capacity of knapsack: M

Maximize $$\sum_{1 \le i \le n} p_i x_i$$

Subject to $$\sum_{1 \le i \le n} w_i x_i \le M$$

$$0 \le x_i \le 1, 1 \le i \le n$$

- **Greedy** approach solves **Fractional Knapsack** problem reasonably in a good time.

Capacity=15

Objects i	1	2	3	4	5	6	7	8
Profit p	10	9	5	8	3	12	9	15
Weight w	3	5	1	2	1	9	6	4
Prof/wt. p/w	3.33	1.8	5	4	3	1.33	1.5	3.75

Sort the array in decreasing order according to the ratio of profit/weight

Objects i	1	2	3	4	5	6	7	8
Prof/wt. p/w	5	4	3.75	3.33	3	1.8	1.5	1.33
Profit p	5	8	15	10	3	9	9	12
Weight w	1	2	4	3	1	5	6	9

Capacity	Total Profit	Profit/weight
15	0	0
15-1= 14	5	5
14-2= 12	5+8=13	4
12-4 = 8	13+15=28	3.75
8- 3 = 5	28+10=38	3.33
5 — 1 = 4	38+ 3= 41	3

Next item has weight 5 but capacity is 4, so we place a fraction of it

0	41 + 1.8*4=48.2	1.8

Greedy strategy applied in fractional knapsack problem

- **Greedy choice**: Take an item or fraction of item with maximum value per unit cost.
- **Optimal substructure**: If we choose a fraction of weight w of the item j, then the remaining weight atmost M-w can be choose from the n-1 item plus $(w_j - w)$ kg of item j.
- Although, both the problems are similar, the fractional knapsack problem is solvable by greedy strategy, but 0-1 knapsack problem are not solvable by greedy algorithm.

The fractional knapsack algorithm

The greedy algorithm:

Step 1: Sort p_i/w_i into non-increasing order.

Step 2: Put the objects into the knapsack according to the sorted sequence as possible as we can.

Example:

$n = 3, M = 20, (p_1, p_2, p_3) = (25, 24, 15)$

$(w_1, w_2, w_3) = (18, 15, 10)$

Sol: $p_1/w_1 = 25/18 = 1.32$

$p_2/w_2 = 24/15 = 1.6$

$p_3/w_3 = 15/10 = 1.5$

Optimal solution: $x_1 = 0, x_2 = 1, x_3 = ½$

- **Greedy** approach solves **Fractional Knapsack** problem reasonably in a good time.

Example: Solving Fractional knapsack problem

- Find the optimal solution for the fractional knapsack problem making use of greedy approach. Consider: n = 5, w = 60 kg and weight of 5 items are $(w_1, w_2, w_3, w_4, w_5) = (5, 10, 15, 22, 25)$ and the corresponding profit is $(p_1, p_2, p_3, p_4, p_5) = (30, 40, 45, 77, 90)$

Solution: Step-01: Compute the value/weight ration for each item.

Items	Weight	Value	Ratio
1	5	30	6
2	10	40	4
3	15	45	3
4	22	77	3.5
5	25	90	3.6

Example: Solving Fractional knapsack problem

- Solution: Step-02: Sort all the items in decreasing order of their value / weight ratio as (6) (4) (3.6) (3.5) (3) for corresponding item **I1 I2 I5 I4 I3**
- Start filling the knapsack by putting the items into it one by one.

Knapsack Weight	Items in Knapsack	Cost
60	Ø	0
55	I1	30
45	I1, I2	70
20	I1, I2, I5	160

Example: Solving Fractional knapsack problem

- Now, Knapsack weight left to be filled is 20 kg but item-4 has a weight of 22 kg.
- Since in fractional knapsack problem, even the fraction of any item can be taken. So, knapsack will contain the following items:

< I1 , I2 , I5 , (20/22) I4 >

Total cost of the knapsack = 160 + (20/27) x 77 = = 160 + 70

- $\Rightarrow$ Total cost of the knapsack= 230 units
- Had the problem been a 0/1 knapsack problem, knapsack would contain the following items-**< I1 , I2 , I5 >** and the knapsack's total cost would be 160 units.

Greedy Algorithm for Fractional Knapsack problem

- Fractional knapsack can be solvable by the greedy strategy
  - Compute the value per pound p_i / w_i for each item
  - Obeying a greedy strategy, take as much as possible of the item with the greatest value per pound.
  - If the supply of that item is exhausted and there is still more room, take as much as possible of the item with the next value per pound, and so forth until there is no more room
  - O(n log n) (we need to sort the items by value per Rupees)
  - Greedy Algorithm?
  - Correctness?

Thanks for Your Attention!

Exercises

Exercises

1. What is optimal substructure?
2. What is greedy strategy?
3. Write briefly about knapsack problem. Explain with an example that greedy algorithm does not work for 0-1 knapsack problem.
4. What is optimal substructure for 0-1 knapsack and fractional knapsack problem?
5. Why is the Knapsack problem NP complete even when it has complexity O(nW)?

Exercises

Solve the following instance of the knapsack problem. There are four items of sizes 2, 3, 5, and 6 and values 3, 4, 5, and 7, and the knapsack capacity is 11.

Solve the following instance of the knapsack problem. There are five items of sizes 3, 5, 7, 8 and 9 and values 4, 6, 7, 9 and 10, and the knapsack capacity is 22.

Exercises

[0/1 Knapsack] Consider the knapsack problem discussed in this section. We add the requirement that $x_i = 1$ or $x_i = 0$, $1 \leq i \leq n$; that is, an object is either included or not included into the knapsack. We wish to solve the problem

$$\max \sum_{1}^{n} p_i x_i$$

$$\text{subject to} \sum_{1}^{n} w_i x_i \leq m$$

$$\text{and} \quad x_i = 0 \text{ or } 1, \quad 1 \leq i \leq n$$

One greedy strategy is to consider the objects in order of nonincreasing density p_i/w_i and add the object into the knapsack if it fits. Show that this strategy doesn't necessarily yield an optimal solution.

Suggested reading

- M.R. Garey and D.S. Johnson, A Guide to the Theory of NP-Completeness, W.H.Freeman and Company, San Francisco, 1979.
- M.R. Garey and D.S. Johnson, Strong NP-Completeness Results: Motivation, Examples, and Implications, *J. Assoc. Comput. Mach.*, **25**(1978), 499–508.