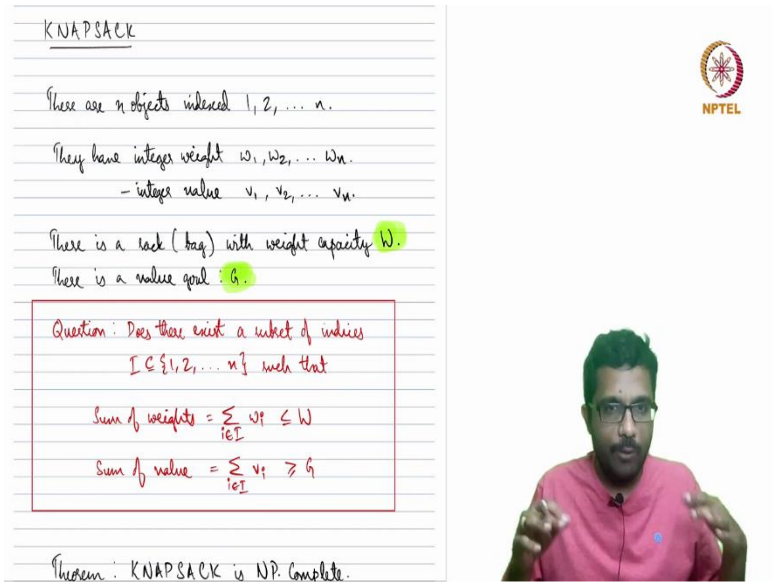
Theory of Computation

Professor Subrahmanyam Kalyanasundaram
Department of Computer Science and Engineering
Indian Institute of Technology, Hyderabad

Lecture 55

Knapsack Problem

(Refer Slide Time: 00:16)



KNAPSACK

There are n objects indexed $1, 2, \dots n$.

They have integer weight $w_1, w_2, \dots w_n$.
- integer value $v_1, v_2, \dots v_n$.

There is a sack (bag) with weight capacity W.
There is a value goal : G.

Question : Does there exist a subset of indices $I \subseteq \{1, 2, \dots n\}$ such that

Sum of weights $= \sum_{i \in I} w_i \leq W$

Sum of value $= \sum_{i \in I} v_i \geq G$

Theorem : KNAPSACK is NP. Complete.

Hello and welcome to lecture 55 of the course Theory of Computation. In this lecture, we will explore yet another NP-complete problem, which is the knapsack problem. In the previous lectures, we have discussed many NP-complete problems, including those based on Boolean formulas like SAT and 3-SAT, as well as problems based on graphs such as CLIQUE, independent set, and vertex cover. In the last lecture, we examined the subset sum problem, which asks whether, given a set of numbers and a target sum, there exists a subset of that set which sums to the target sum. All these problems are NP-complete.

To remind ourselves, NP-complete problems are considered to be the hardest problems in the class NP. It is widely believed that they do not have a polynomial-time algorithm. If we could show that any NP-complete problem has a polynomial-time algorithm, we would prove that P equals NP, which is a major open question that has been unresolved for 50 years. It is widely expected that P is not equal to NP, and hence, most people do not think that we will be able to provide polynomial-time algorithms for these NP-complete problems.

This lecture is going to be brief. We will explore another problem called knapsack. Let's start by understanding the problem. We have n objects indexed from 1 to n, and each object has a weight, denoted as w1, w2, ..., wn. The first object has weight w1, the second object has weight w2, and so on. Each object also has a value, denoted as v1, v2, ..., vn. So, there is both a weight and a value associated with each object. For instance, gold might be very valuable despite its small weight, whereas iron might not be as valuable for the same weight.

The scenario is as follows: you are a thief or a robber, and you are robbing a place with a large assortment of objects. You have brought a sack or a bag with you, which is what the knapsack refers to. This bag can carry objects but has limited capacity; it can only hold items up to a certain weight. As a thief, you want to maximize the value of the items you carry because that is how you can sell them for the most money, given the risks you are taking. Therefore, you want to maximize the value of the items in your bag. However, the bag has a physical limit and cannot hold items exceeding a certain total weight.

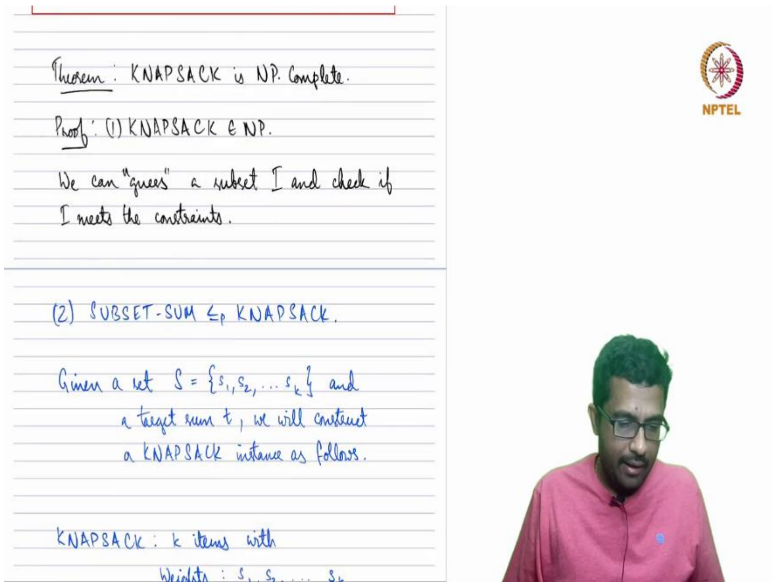
The goal is to pack as many valuable items as possible such that the total weight does not exceed the bag's capacity. We will now formulate this question as a decision problem.

There are n items indexed from 1 to n, each with a weight w1 to wn and a value v1 to vn. Given the capacity of the bag W (capital W) and a goal value G, the task is to determine if there exists a subset of items whose total weight is less than or equal to W and whose total value is at least G. In other words, does there exist a subset of items such that the sum of their weights is below the capacity W and the sum of their values exceeds or meets the goal G?

To summarize the problem statement:There are n items, each with a specific weight and value.The bag can carry a maximum weight of W.The goal is to achieve at least a total value of G from the items packed in the bag.We need to determine if there exists a subset of items meeting these conditions.

This formulation allows us to understand the knapsack problem as a decision problem and recognize its complexity and its place among NP-complete problems.

(Refer Slide Time: 05:15)



Theorem: KNAPSACK is NP Complete.

Proof: (1) KNAPSACK $\in$ NP.

We can "guess" a subset I and check if I meets the constraints.

(2) SUBSET-SUM $\leq_p$ KNAPSACK.

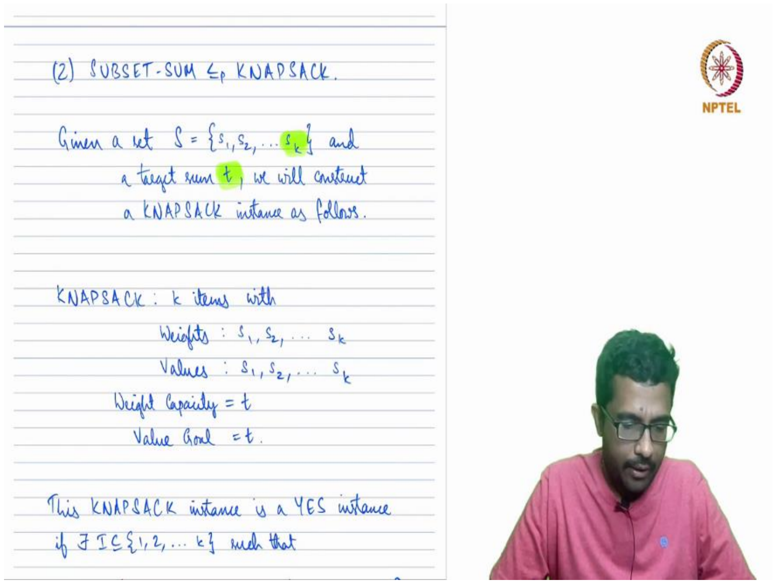
Given a set $S = \{s_1, s_2, \dots s_k\}$ and a target sum t, we will construct a KNAPSACK instance as follows.

KNAPSACK: k items with

Weights: $s_1, s_2, \dots s_k$

This is a natural desire: to obtain items with weights below capacity but values above a given goal. The question is about selecting items with a maximum capacity WWW and values meeting or exceeding the goal, which is an NP-complete problem. It's akin to the subset sum dilemma where we aim to choose a subset of terms or numbers whose sum matches a target. Similarly, here, we seek a subset of items that satisfy both conditions. This problem is also NP-complete. Initially, it's in NP due to the guess-and-verify method. We guess a subset by selecting or not selecting each element, then verify if the subset meets the weight and value constraints. If it's truly a knapsack problem, one of these guesses will be correct.

(Refer Slide Time: 05:53)



(2) SUBSET-SUM $\leq_p$ KNAPSACK.

Given a set $S = \{s_1, s_2, \dots s_k\}$ and a target sum t, we will construct a KNAPSACK instance as follows.

KNAPSACK: k items with

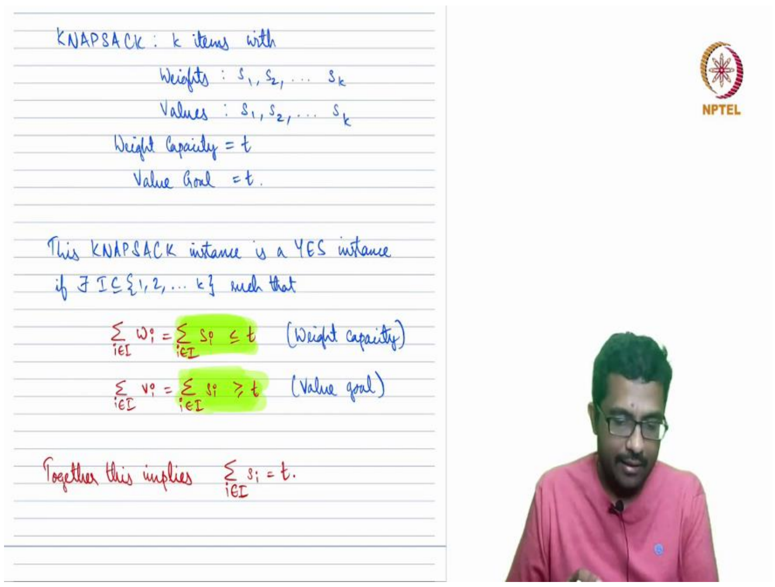
Weights: $s_1, s_2, \dots s_k$

Values: $s_1, s_2, \dots s_k$

Weight Capacity $= t$

Value Goal $= t$.

This KNAPSACK instance is a YES instance if $\exists\, I \subseteq \{1, 2, \dots k\}$ such that


KNAPSACK : k items with

Weights : $s_1, s_2, \dots s_k$

Values : $s_1, s_2, \dots s_k$

Weight Capacity $= t$

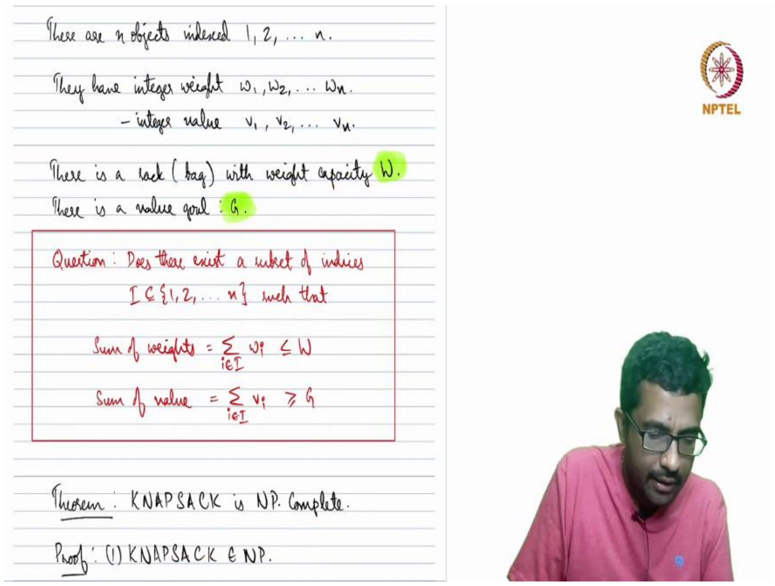
Value Goal $= t$.

This KNAPSACK instance is a YES instance if $\exists\, I \subseteq \{1, 2, \dots k\}$ such that

$$\sum_{i \in I} w_i = \sum_{i \in I} s_i \leq t \quad \text{(Weight capacity)}$$

$$\sum_{i \in I} v_i = \sum_{i \in I} s_i \geq t \quad \text{(Value goal)}$$

Together this implies $\sum_{i \in I} s_i = t$.


There are n objects indexed $1, 2, \dots n$.

They have integer weight $w_1, w_2, \dots w_n$.

– integer value $v_1, v_2, \dots v_n$.

There is a sack (bag) with weight capacity W.

There is a value goal : G.

Question : Does there exist a subset of indices $I \subseteq \{1, 2, \dots n\}$ such that

$$\text{Sum of weights} = \sum_{i \in I} w_i \leq W$$

$$\text{Sum of value} = \sum_{i \in I} v_i \geq G$$

Theorem : KNAPSACK is NP Complete.

Proof : (1) KNAPSACK $\in$ NP.

To prove NP-completeness, we need to show that some NP-complete language reduces to this problem. We'll reduce the subset sum problem to the knapsack problem, as they are similar in nature. In subset sum, we choose a subset of numbers that sum to a target. Similarly, in the knapsack problem, we choose a subset of items. This similarity is useful when proving a language is NP-complete, as we can look for similar NP-complete problems for reduction.

Subset sum asks if there is a subset of a set S (with elements s1 to sk) that sums to a target t. To construct a knapsack instance from a subset sum instance, we need to define n items, their weights, values, weight capacity, and value goal.

Here’s how we define the knapsack instance:We have k items (same as the number of elements in the subset sum problem).The weights of the items are s1 to sk (the same numbers from the subset sum problem).The values of the items are also s1 to sk.The weight capacity is t (the target sum from the subset sum problem).The value goal is also t.

This reduction is straightforward: the numbers from the subset sum problem become both the weights and values in the knapsack problem, and the target sum becomes both the weight capacity and the value goal. This takes linear time to write down, making it a polynomial-time reduction.

The correspondence is clear: if the knapsack instance is a yes instance (meaning there is a subset of items whose total weight is at most t and whose total value is at least t), then the subset sum instance is also a yes instance (there is a subset of numbers summing to t). Hence, if the subset sum instance is a yes instance, then the knapsack instance will also be a yes instance, and vice versa.

For a yes instance of knapsack, we need to satisfy two conditions: the sum of the weights must be at most the weight capacity, and the sum of the values must be at least the goal. Here, the weight capacity and the goal are both t. The weights are s1 to sk, and the values are also s1 to sk.

So, we need to check if there is a subset of items (indices) such that the sum of their weights is at most t. This means we need to find a subset I of indices such that the sum of si for all i in I is at most t. This is the weight constraint.

Similarly, for the value goal, we need to check if the sum of the values of the selected items is at least t. Since the values are also s1 to sk, this means we need the sum of si for all i in I to be at least t.

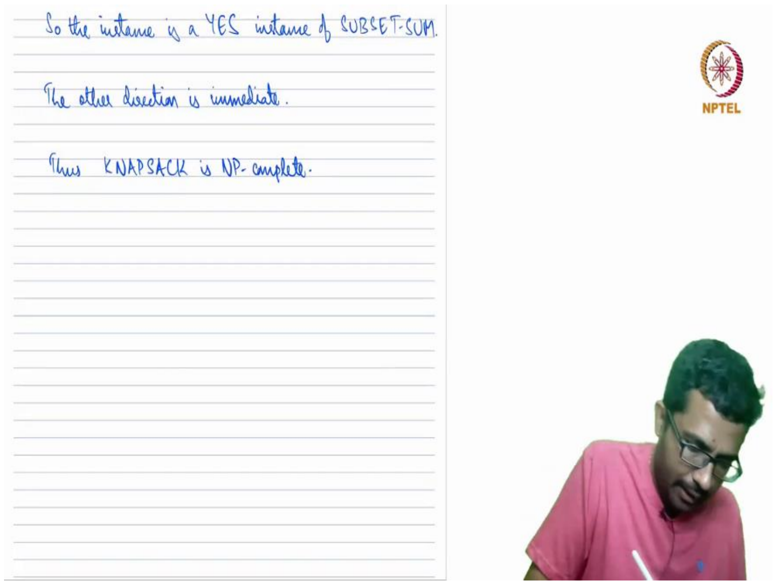
Therefore, we have two inequalities:The sum of the selected weights (sum of si for i in I) should be at most t.The sum of the selected values (sum of si for i in I) should be at least t.

Combining these inequalities, if the knapsack instance is a yes instance, both conditions must be satisfied. This implies that the sum of the selected si is exactly equal to t.

This condition is precisely what we require for a yes instance of subset sum. We need a subset of the given set S that sums to the target sum t. Therefore, a yes instance of knapsack implies a yes instance of subset sum. Similarly, a yes instance of subset sum implies a yes instance of knapsack, as both directions satisfy the same condition of summing to t.

Hence, the reduction from subset sum to knapsack is valid, and the knapsack problem is NP-complete.

(Refer Slide Time: 11:59)



So the instance is a YES instance of SUBSET-SUM.

The other direction is immediate.

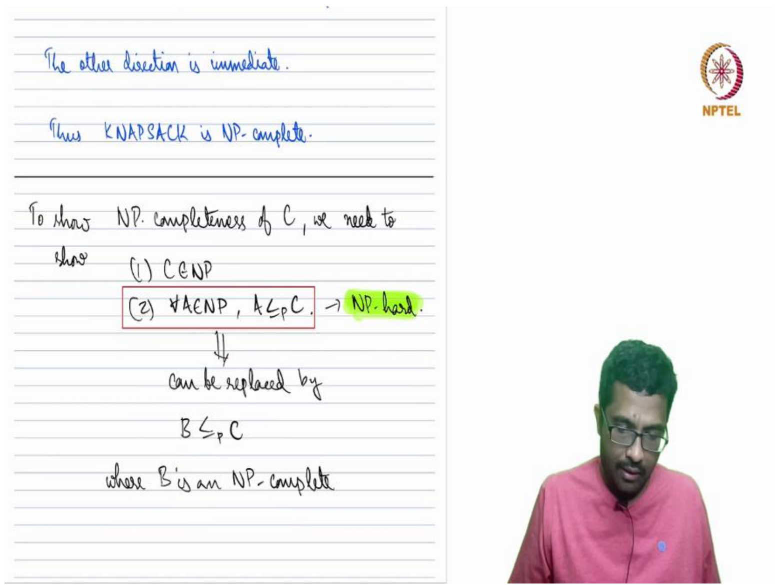
Thus KNAPSACK is NP-complete.

So, this means that the correspondence is also proved. A yes instance of knapsack implies a yes instance of subset sum, and vice versa. This shows that we have a yes instance of the knapsack if and only if we have a yes instance of subset sum. Together, this implies that the knapsack problem is NP-complete. It's a brief proof, but that’s all there is to it.

In the knapsack problem, we have weights and values for each item. We reduced subset sum to knapsack by making the weights and values equal to the numbers in the subset sum, and setting both the weight capacity and value goal equal to the target sum. If the knapsack is a yes instance, it gives us a subset whose sum equals the target sum. If there is a subset with the target sum, the same subset will be a yes instance of the knapsack. It's straightforward.

This reduction shows that subset sum reduces to knapsack, and since the reduction takes polynomial time, it proves that knapsack is NP-complete. Now, one small thing I want to mention.

(Refer Slide Time: 13:38)



The other direction is immediate.

Thus KNAPSACK is NP-complete.

To show NP completeness of C, we need to show

(1) $C \in NP$

$\boxed{(2)\ \forall A \in NP,\ A \leq_p C.}$ → NP-hard.

⇓

can be replaced by

$B \leq_p C$

where B is an NP-complete

So, there are 2 parts in showing NP completeness. To show NP completeness of, let's say C, we need to show 2 things: one is that C is in NP, and two is that for all A in NP, A reduces to C in polynomial time. This second condition can actually be replaced by showing that B reduces to C in polynomial time, where B is an NP-complete problem. Instead of showing that all A in NP reduce to C, we just show that a selected NP-complete problem reduces to C. This is the strategy we used by reducing subset sum, already shown to be NP-complete, to knapsack.

The second part alone is usually referred to as NP-hard or NP-hardness. To show NP completeness, we need to show that C is in NP (condition 1) and that C is NP-hard (condition 2). NP-hard means all languages in NP reduce to C, or equivalently, some NP-complete language reduces to C. Sometimes, the second condition is referred to as NP-hardness.

I wanted to stress this terminology because, as students of a Theory of Computation course or when learning computational complexity, you may come across the term NP-hardness. NP-hardness means that all languages in NP reduce to that language. To show something is NP-complete, we need to show membership in NP (condition 1) and NP-hardness (condition 2). These two together show that a language is NP-complete.

So, that concludes lecture number 55. We defined the knapsack problem, a decision problem, and showed it is NP-complete by reducing it from subset sum. We also briefly discussed NP-hardness. That completes lecture 55. In the next lecture, lecture 56, we will see yet another NP-complete problem. See you in lecture 56. Thank you.