Learning Goals

- State the Subset Sum problem
- Understand the reduction from 3-EXACT COVER to Subset Sum
- State the Knapsack problem
- Understand the reduction from Subset Sum to Knapsack

Subset Sum

Definition (Subset Sum)

In the Subset Sum problem, we are given nonnegative integers $w_1, \ldots, w_n$, and a target number W, and we must decide whether there is a subset of $\{w_1, \ldots, w_n\}$ that adds up to precisely W.

Subset Sum

Definition (Subset Sum)

In the Subset Sum problem, we are given nonnegative integers $w_1, \ldots, w_n$, and a target number W, and we must decide whether there is a subset of $\{w_1, \ldots, w_n\}$ that adds up to precisely W.

- Subset Sum $\in$ NP.

Subset Sum

Definition (Subset Sum)

In the Subset Sum problem, we are given nonnegative integers $w_1, \dots, w_n$, and a target number W, and we must decide whether there is a subset of $\{w_1, \dots, w_n\}$ that adds up to precisely W.

- Subset Sum $\in$ NP.
  - Certificate: a subset of $\{w_1, \dots, w_n\}$;
  - Verifier: Check the sum of the subset.

Subset Sum

Definition (Subset Sum)

In the Subset Sum problem, we are given nonnegative integers $w_1, \ldots, w_n$, and a target number W, and we must decide whether there is a subset of $\{w_1, \ldots, w_n\}$ that adds up to precisely W.

- Subset Sum $\in$ NP.
  - Certificate: a subset of $\{w_1, \ldots, w_n\}$;
  - Verifier: Check the sum of the subset.
- Subset Sum problem can be solved by a dynamic programming that runs in time $O(nW)$.

Dynamic Programming for Subset Sum

- Set up boolean array $S[0 \dots W]$, itialized to $W[0] = \text{True}$, $W[j] = FALSE$ for $j = 1, \cdots, W$.
- For $i = 1, \dots, n$
  - For $j = W, W - 1, \cdots, 0$, if $S[j] = \text{True}$, set $S[j + w_i] = \text{True}$.
- Return $S[W]$.

Dynamic Programming for Subset Sum

- Set up boolean array $S[0 \ldots W]$, itialized to $W[0] = \text{True}$, $W[j] = FALSE$ for $j = 1, \cdots, W$.
- For $i = 1, \ldots, n$
  - For $j = W, W-1, \cdots, 0$, if $S[j] = \text{True}$, set $S[j + w_i] = \text{True}$.
- Return $S[W]$.

Question

Is this a polynomial-time algorithm?

Dynamic Programming for Subset Sum

- Set up boolean array $S[0 \dots W]$, itialized to $W[0] = \text{True}$, $W[j] = FALSE$ for $j = 1, \cdots, W$.
- For $i = 1, \dots, n$
  - For $j = W, W-1, \cdots, 0$, if $S[j] = \text{True}$, set $S[j + w_i] = \text{True}$.
- Return $S[W]$.

Question

Is this a polynomial-time algorithm?

Answer.

- Input length: $n + \log W$;
- Running time: $O(nW) = O(n \exp(\log W))$;
- Not in polynomial time, but in *pseudo-polynomial* time.

□

NP-completeness of Subset Sum

Proposition

3-EXACT COVER $\leq_P$ Subset Sum.

NP-completeness of Subset Sum

Proposition

3-EXACT COVER $\leq_P$ Subset Sum.

Proof.

Given 3-EXACT COVER instance with universe $U = \{u_1, \ldots, u_n\}$, and $S_1, \cdots, S_m \subseteq U$,

NP-completeness of Subset Sum

Proposition

3-EXACT COVER $\leq_P$ Subset Sum.

Proof.

Given 3-EXACT COVER instance with universe $U = \{u_1, \ldots, u_n\}$, and $S_1, \cdots, S_m \subseteq U$, construct the following subset instance:

- For each S_i, construct $w_i = \sum_{j:u_j \in S_i} (1+m)^{j-1}$;
- $W = \sum_{j=1}^{n} (1+m)^{j-1}$.

NP-completeness of Subset Sum

Proposition

3-EXACT COVER $\leq_P$ Subset Sum.

Proof.

Given 3-EXACT COVER instance with universe $U = \{u_1, \ldots, u_n\}$, and $S_1, \cdots, S_m \subseteq U$, construct the following subset instance:

- For each S_i, construct $w_i = \sum_{j:u_j \in S_i} (1+m)^{j-1}$;
- $W = \sum_{j=1}^{n} (1+m)^{j-1}$.

The 3-EXACT COVER instance has answer YES if and only if the Subset Sum instance has answer YES. $\square$

NP-completeness of Subset Sum

Proposition

3-EXACT COVER $\leq_P$ Subset Sum.

Proof.

Given 3-EXACT COVER instance with universe $U = \{u_1, \ldots, u_n\}$, and $S_1, \cdots, S_m \subseteq U$, construct the following subset instance:

- For each S_i, construct $w_i = \sum_{j: u_j \in S_i} (1+m)^{j-1}$;
- $W = \sum_{j=1}^{n} (1+m)^{j-1}$.

The 3-EXACT COVER instance has answer YES if and only if the Subset Sum instance has answer YES. □

Key idea: Use $(m+1)$-ary representation of integers to prevent carries in addition.

Polynomial time vs. Pseudo-polynomial time

Question

In the Subset Sum problem, if W is bounded by a polynomial function of n, then does the problem have a polynomial-time algorithm?

Polynomial time vs. Pseudo-polynomial time

Question

In the Subset Sum problem, if W is bounded by a polynomial function of n, then does the problem have a polynomial-time algorithm?

Answer.

Yes. The runtime of the DP algorithm is now $O(nW) = O(\text{poly}(n))$. $\square$

Polynomial time vs. Pseudo-polynomial time

Question

Does Subset Sum admit a polynomial time algorithm if W in the input is in *unary* representation?

In unary representation, an integer k is represented as k one's.

Polynomial time vs. Pseudo-polynomial time

Question

Does Subset Sum admit a polynomial time algorithm if W in the input is in *unary* representation?

In unary representation, an integer k is represented as k one's.

Answer.

Yes. Because the input length is now $W + n$, instead of $n + \log W$. $\square$

Polynomial time vs. Pseudo-polynomial time

Question

Does the following problem admit a polynomial-time algorithm? Given a graph G that is not connected, and a number k, does there exist a subset of its connected components whose union has size exactly k?

Polynomial time vs. Pseudo-polynomial time

Question

Does the following problem admit a polynomial-time algorithm? Given a graph G that is not connected, and a number k, does there exist a subset of its connected components whose union has size exactly k?

Proof.

Answer Yes. $k \leq$ number of nodes of G. If there are n nodes, the dynamic programming runs in time $O(n^2)$. □

The Knapsack Problem

Definition

In the Knapsack problem, we are given a knapsack with capacity C and n item, where item i has weight w_i and value v_i. We are also given a target value W. We must decide whether there is a subset of the items whose total weight is no more than C and whose total value is no less than W.

The Knapsack Problem

Definition

In the Knapsack problem, we are given a knapsack with capacity C and n item, where item i has weight w_i and value v_i. We are also given a target value W. We must decide whether there is a subset of the items whose total weight is no more than C and whose total value is no less than W.

Claim (Exercise)

Knapsack is NP-complete.

Categories of basic NP-complete problems

- Binary decision: 3-SAT
- Packing problems: Independent Set
- Covering problems: Vertex cover, Set Cover
- Sequencing problems: Hamiltonian Cycle/Path, Traveling Salesman Problem
- Partitioning problems: 3-Dimensional Matching, 3-Coloring, 3-Exact Cover
- Numerical problems: Subset Sum, Knapsack