

Subset-Sum

- Recall the language/decision problem SUBSET-SUM:

Instance: $s_1, \dots, s_n, t \in \mathbb{Z}$

Question: Does there exist $S \subseteq \{1, \dots, n\}$ such that $\sum_{i \in S} s_i = t$?

- Theorem:** SUBSET-SUM is **NP**-complete.
- Proof: See textbook pages 268-271.

An algorithm for Subset-Sum

- Let s_1, \dots, s_n, t be an instance of SUBSET-SUM.
- Define $A[i, j]$ as follows:

$$A[i, j] = \begin{cases} 1, & \text{if } \exists S \subseteq \{1, \dots, i\} \text{ such that } \sum_{k \in S} s_k = j \\ 0, & \text{otherwise} \end{cases}$$

- If we knew the value of $A[n, t]$ then we would know the answer for this instance.
- Below is a recurrence expressing $A[i, j]$ in terms of “smaller” entries in the table.

$$A[i, j] = \begin{cases} \max\{A[i-1, j], A[i-1, j-s_i]\}, & \text{if } i, j > 0 \\ 1, & \text{if } j = 0 \\ 0, & \text{if } i = 0 \text{ and } j \neq 0 \end{cases}$$

The idea is that if $A[i, j] = 1$ then either s_i is in the set summing to j , or it isn't. If it is, then the remaining elements in the set add up to $j - s_i$ and are a subset of $\{1, \dots, i-1\}$, so we will find $A[i-1, j-s_i] = 1$. If s_i isn't in the set summing to j , then we will find $A[i-1, j] = 1$.

- The algorithm just fills in the values of $A[i, j]$ in a table: on input $i \langle s_1, \dots, s_n, t \rangle \dots$
 - for $i \leftarrow 0, \dots, n$ do
 - for $j \leftarrow 0, \dots, t$ do
 - compute $A[i, j]$ using recurrence
 - accept iff $A[n, t] = 1$

Step 3 takes constant time if we have random access to the array A , or $O(nt)$ time otherwise (let's assume random access to make things easier). So the running time is $O(nt)$. Is this polynomial time? No: if t is represented in binary (which is a sensible way to represent numbers), then it requires only $O(\log t)$ input bits. Consider, for example, the case where

every s_i and t are $\Theta(2^n)$, where n is the number of s_i 's. Then the input size is $O(n^2)$, since there are $O(n)$ numbers and each takes $O(n)$ bits to represent in binary. But the running time is $\Omega(nt) = \Omega(n2^n) = \Omega(\sqrt{N}2^{\sqrt{N}})$, where $N = n^2$ is the input size.

- However, consider the problem UNARY-SUBSET-SUM, which is identical to SUBSET-SUM except that the numbers are represented in unary (the unary representation of $k \in \mathbb{N}$ is just 1^k , i.e. a sequence of k 1's). Then UNARY-SUBSET-SUM is in \mathbf{P} , since the input size is $\Omega(n+t)$ and thus the running time $O(nt)$ is quadratic in the input size. In fact, it is only necessary for t to be represented in unary for the above to hold.
- An \mathbf{NP} -complete language is called “weakly \mathbf{NP} -complete” if it has in its description one or more integer parameters and the corresponding language where these parameters are represented in unary is in \mathbf{P} . So SUBSET-SUM is weakly \mathbf{NP} -complete.
- For contrast, note that CLIQUE is not weakly \mathbf{NP} -complete, assuming $\mathbf{P} \neq \mathbf{NP}$ (i.e. it is strongly \mathbf{NP} -complete): if the parameter representing the clique size is written in unary the language is still \mathbf{NP} -complete.
- Notice that in the reduction $3\text{SAT} \leq_p \text{SUBSET-SUM}$, the numbers that the reduction outputs are extremely large (i.e. even for say 10 variables and 10 clauses the numbers have 20 digits, i.e. they are of the order 10^{20}). In retrospect, we can see why this is necessary: if the numbers are small then we can solve SUBSET-SUM quickly, so if there is a reduction from 3SAT to instances of SUBSET-SUM with small numbers (i.e. polynomial in the size of the formula), then we could solve 3SAT in polynomial time, implying $\mathbf{P} = \mathbf{NP}$.