SUBSET-SUM is NP-Complete

- The SUBSET-SUM problem:
  - Instance: We are given a set *S* of positive integers, and a target integer *t.*
  - Question: does there exist a subset of *S* adding up to *t?*
    - Example: {1, 3, 5, 17, 42, 391}, target 50
      - The subset sum problem is a good problem to use when proving NP-completeness for problems defined on sets of integers.
  - We will show that 3-SAT $\leq_P$ SUBSET-SUM.
    - So: We are given an arbitrary 3-SAT formula and we wish to derive a set *S* of integers and a target integer t.
      - Then we prove that 3-SAT is satisfiable iff a subset of S adds up to t.

3-SAT $=_P$ SUBSET-SUM

- Idea: we use "bit descriptions" that essentially describe the construction of the 3-SAT formula.
  - The integers in S will be derived from this description.
  - As before, we assume that there are *m* Boolean variables and *n* clauses.
  - Our description uses bit fields that represent different aspects of the 3-SAT formula: it has two lines for each logic variable:
    - One line specifies which clauses use the true version of a variable.
    - Another line describes which clauses use the false version of the variable.

– For every variable u_i we create a line T_i corresponding to the true value of u_i and another line F_i for the false value of u_i as follows:

	u_1	u_2	...	u_i	...	u_m	c_1	c_2	...	c_n
...	...	...	...	...	...	...	...	...	...	...
T_i	0	0	0	1	0	0	.	.	.	.
F_i	0	0	0	1	0	0	.	.	.	.
...	...	...	...	...	...	...	...	...	...	...

There is a 1 under c_j in row T_i iff the j^{th} clause contains u_i.
(Similarly for the F_i row).

– Example description for:

$(u_1 \vee \neg u_3 \vee \neg u_4) \wedge (\neg u_1 \vee u_2 \vee \neg u_4)$

	u_1	u_2	u_3	u_4	c_1	c_2
T_1	1	0	0	0	1	0
F_1	1	0	0	0	0	1
T_2	0	1	0	0	0	1
F_2	0	1	0	0	0	0
T_3	0	0	1	0	0	0
F_3	0	0	1	0	1	0
T_4	0	0	0	1	0	0
F_4	0	0	0	1	1	1

- We will get the numbers for *S* by considering the rows to be integers.
- When adding the integers we do not want any complicating carry-over into the next column so we consider the entries to be numbers in a higher base.
  - A base corresponding to a three bit integer would do (an octal number) but for simplicity, each number is considered to be a base-10 digit.
- The selection of the required subset from S will designate a subset of lines from the description.
- This selection must be forced to choose either a T_i line or an F_i line (not both) for each value of i.
  - This means that the target integer t will start with m 1's.

- What about the sums of entries in the columns for the clauses?
  - Looking at the description lines we see that the sum in a column could be 1, 2, or 3.
  - We want the target number to have a specific value so we append more lines (integers) to the array to give the selection mechanism a chance to get a subset with a specific target sum.
    - We will show that this does not destroy our ability to do an appropriate selection from the T_i, F_i rows.
  - For every clause column, we need two more integers: one with a 1 and another with a 2 in that column and 0 everywhere else
  - Make the target have a 4 in the digits for the clause columns.
    - Note: For any column, the nonzero digits in all integers add up to at most 6 (so our base-10 simplification will never see a carry-over).

– Going back to our example:

$(u_1 \vee \neg u_3 \vee \neg u_4) \wedge (\neg u_1 \vee u_2 \vee \neg u_4)$

	u_1	u_2	u_3	u_4	c_1	c_2
T_1	1	0	0	0	1	0
F_1	1	0	0	0	0	1
T_2	0	1	0	0	0	1
F_2	0	1	0	0	0	0
T_3	0	0	1	0	0	0
F_3	0	0	1	0	1	0
T_4	0	0	0	1	0	0
F_4	0	0	0	1	1	1
$S1_1$	0	0	0	0	1	0
$S2_1$	0	0	0	0	2	0
$S1_2$	0	0	0	0	0	1
$S2_2$	0	0	0	0	0	2
Target :	1	1	1	1	4	4

- Suppose the formula were satisfiable:
  - We need to show that there is a subset of *S* with target sum t.
  - Choose integer from the T_i, F_i rows corresponding to true literals, giving sums of 1 in the literal-digit positions.
  - Using only the T_i, F_i rows, we get sums that are 1, 2, or 3 in the clause columns.
  - Choose appropriate “slack integers” (from the $S1_i$ and $S2_i$ rows to make those sums equal to 4).
  - The final sum matches the target in all digits, as required.

- Suppose a set of integers sum to the target:
  - We need to show that we can get a satisfying assignment of the given 3-SAT formula.
  - There must be exactly one integer (i.e. row) selected from each pair of the T_i, F_i rows, or there is no way to get a 1 in the initial columns of the target.
  - The selection thus defines the obvious assignment to variables, but is it a satisfying assignment?
  - For each clause, the "slack integers" in the chosen set can only add up to at most 3 in that clause column.
  - There must be at least one 1 contributed from some integer corresponding to the T_i, F_i rows.
  - That corresponds to a true literal in that clause and the formula is satisfied.

- Finishing our Proof that SUBSET-SUM is NP-Complete:
  - Since 3-SAT is NP-complete, we have just demonstrated that SUBSET-SUM is NP-hard.
  - But is it in NP? Yes, because:
    - We have a certificate: the subset achieving the target sum.
    - A verification algorithm would verify that the numbers specify a subset and furthermore that they add up to the target.
  - Finally:
    - SUBSET-SUM is NP-hard + SUBSET-SUM in NP ➔ SUBSET-SUM is NP-complete.

Coin-Changing is NP-Hard

- Prove by reduction from SUBSET-SUM.
  - We want to prove:
    SUBSET-SUM $\leq_P$ COIN-CHANGING.
  - We are given an arbitrary instance of subset-sum:
    - $s_1, s_2, \ldots, s_n$ and a target t.
  - We want to create a particular coin changing problem by specifying:
    - the number of coin denominations, the value of each denomination and a payout value S.
      - Recall that each coin can be used more than once in the payout.
  - Let M be equal to $\max\{s_1, s_2, \ldots, s_n\}$.
  - For every number s_i create two coins C_i and N_i:
    - We use a strategy that is similar to our previous proof, but most of the numbers in the array are in "base-2n" instead of base-10.
    - The first column is special and will work in base-2nM.

	$base-2nM$	$base-2n$	...	$base-2n$	...	$base-2n$
		s_1	...	s_i	...	s_n
...	...	...	...	...	...	...
C_i	s_i	0	...	1	...	0
N_i	0	0	...	1	...	0
...	...	...	...	...	...	...

    - Intuition:
      - Choosing coin C_i means putting s_i into the subset.
      - Choosing coin N_i means <u>not</u> putting s_i into the subset.
      - Our payout value S will be the target sum t.
      - The column sums will be: t for the first column with 1 for all the others.

– Show:
  - If there exists a subset with sum *t* then it is possible to pay out the sum with $\leq$ n coins.

– Proof:
  - If s_i is chosen then take coin C_i, otherwise take coin N_i.
  - So, the sum in the first column is t as required and all other columns sum to 1.

– Show:
  - If it is possible to pay out the sum *S* with $\leq$ *n* coins, then there exists a subset with sum *t.*

– Proof:
  - Since there are $\leq$ n coins, there is never a carry-over into an adjacent column (consider the base of the entries).
  - So from every pair C_i, N_i exactly one is chosen.
    – This is the only possibility that will ensure that the column sums are all ones after the first column.
  - We construct a subset of the s_1, s_2,…, s_n as follows: s_i is selected to be in the subset iff C_i was chosen.
    – We will get a first column sum of *t*.

- Since SUBSET-SUM is NP-complete and since we have just shown that SUBSET-SUM $\leq_P$ COIN-CHANGING we can deduce that COIN-CHANGING is NP-hard.

- As a certificate we can use the set of coins paying out the sum.
- For verification we could:
  - Check to verify that the certificate contains only valid coins.
  - Check that the number of coins is $\leq n$.
  - Check the sum.

- So:
  - COIN-CHANGING is in NP.

- Finally:
  - COIN-CHANGING is NP-complete.