Fundamental Computer Science
Lecture 4: Complexity
Pseudo-polynomial algorithms

Denis Trystram
MoSIG1 and M1Info – University Grenoble-Alpes

March, 2021

Content

- Deal with numerical problems
- Refine the notion of NP-completeness
  - pseudo-polynomial problems
  - weakly and strongly NP-hardness

SubsetSum

Let us introduce a new problem:

SubsetSum

Input: a set of positive integers $A = \{a_1, a_2, \ldots, a_k\}$
$t \in \mathbb{N}$

Question: is there a set $B \subseteq A$ such that $\sum_{a_i \in B} a_i = t$?

SubsetSum $\in$ NP-complete

First, SubsetSum $\in \mathcal{NP}$

Verifier

- given the set $B \subseteq A$, create the sum of the elements in B and compare with t

SubsetSum $\in$ NP-complete

First, SubsetSum $\in \mathcal{NP}$

Verifier

- given the set $B \subseteq A$, create the sum of the elements in B and compare with t

We will show next:
3SAT $\leq_P$ SubsetSum

Construction of the reduction from $3\mathrm{SAT}$

1. for each variable x_i create two decimal numbers y_i and z_i
    - intuition:
      – select one of y_i, z_i in B
      – if y_i is in B, then $x_i = \mathrm{TRUE}$
      – if z_i is in B, then $x_i = \mathrm{FALSE}$

Construction of the reduction from $3\mathrm{SAT}$

1. for each variable x_i create two decimal numbers y_i and z_i
   - intuition:
     – select one of y_i, z_i in B
     – if y_i is in B, then $x_i = \mathrm{TRUE}$
     – if z_i is in B, then $x_i = \mathrm{FALSE}$
   - each y_i, z_i has two parts:
     – a variable part (see above)
     – another part built from the clause it appears in
2. for each clause C_j, we create two decimal numbers g_j and h_j

	x_1	x_2	x_3	$\dots$	x_n	C_1	C_2	$\dots$	C_m
y_1	1	0	0	$\dots$	0				
z_1	1	0	0	$\dots$	0				
y_2	0	1	0	$\dots$	0				
z_2	0	1	0	$\dots$	0				
y_3	0	0	1	$\dots$	0				
z_3	0	0	1	$\dots$	0				
$\vdots$	$\vdots$	$\vdots$		$\ddots$	$\vdots$				
y_n	0	0	0	0	1				
z_n	0	0	0	0	1				
g_1						1	0	$\dots$	0
h_1						2	0	$\dots$	0
g_2						0	1	$\dots$	0
h_2						0	2	$\dots$	0
$\vdots$						$\vdots$		$\ddots$	$\vdots$
g_m						0	0	0	1
h_m						0	0	0	2

	x_1	x_2	x_3	$\dots$	x_n	C_1	C_2	$\dots$	C_m
y_1	1	0	0	$\dots$	0	1	0	$\dots$	0
z_1	1	0	0	$\dots$	0	0	0	$\dots$	1
y_2	0	1	0	$\dots$	0	0	1	$\dots$	0
z_2	0	1	0	$\dots$	0	1	0	$\dots$	0
y_3	0	0	1	$\dots$	0	1	1	$\dots$	0
z_3	0	0	1	$\dots$	0	0	0	$\dots$	1
$\vdots$	$\vdots$	$\vdots$		$\ddots$	$\vdots$	$\vdots$		$\ddots$	$\vdots$
y_n	0	0	0	0	1	0	0	$\dots$	1
z_n	0	0	0	0	1	0	0	$\dots$	0
g_1						1	0	$\dots$	0
h_1						2	0	$\dots$	0
g_2						0	1	$\dots$	0
h_2						0	2	$\dots$	0
$\vdots$						$\vdots$		$\ddots$	$\vdots$
g_m						0	0	0	1
h_m						0	0	0	2

	x_1	x_2	x_3	$\dots$	x_n	C_1	C_2	$\dots$	C_m
y_1	1	0	0	$\dots$	0	1	0	$\dots$	0
z_1	1	0	0	$\dots$	0	0	0	$\dots$	1
y_2	0	1	0	$\dots$	0	0	1	$\dots$	0
z_2	0	1	0	$\dots$	0	1	0	$\dots$	0
y_3	0	0	1	$\dots$	0	1	1	$\dots$	0
z_3	0	0	1	$\dots$	0	0	0	$\dots$	1
$\vdots$	$\vdots$	$\vdots$		$\ddots$	$\vdots$	$\vdots$		$\ddots$	$\vdots$
y_n	0	0	0	0	1	0	0	$\dots$	1
z_n	0	0	0	0	1	0	0	$\dots$	0
g_1						1	0	$\dots$	0
h_1						2	0	$\dots$	0
g_2						0	1	$\dots$	0
h_2						0	2	$\dots$	0
$\vdots$						$\vdots$		$\ddots$	$\vdots$
g_m						0	0	0	1
h_m						0	0	0	2
t	1	1	1	$\dots$	1	4	4	$\dots$	4

Example

$$(x_1 \vee \bar{x}_2 \vee x_3) \wedge (x_2 \vee x_3 \vee x_4) \wedge (\bar{x}_1 \vee \bar{x}_3 \vee x_4)$$

	x_1	x_2	x_3	x_4	C_1	C_2	C_3
y_1	1	0	0	0	1	0	0
z_1	1	0	0	0	0	0	1
y_2		1	0	0	0	1	0
z_2		1	0	0	1	0	0
y_3			1	0	1	1	0
z_3			1	0	0	0	1
y_4				1	0	1	1
z_4				1	0	0	0
g_1					1	0	0
h_1					2	0	0
g_2						1	0
h_2						2	0
g_3							1
h_3							2
W	1	1	1	1	4	4	4

SubsetSum $\in$ NP-complete

- Size of the created instance:
  - $|A| = 2n + 2m$
  - each created integer has at most $n + m$ digits (including t)
    $\rightarrow$ integers in the interval $[0, 10^{n+m}]$
    $\rightarrow$ binary representation: at most $\log_2 10^{n+m} = O(n + m)$ bits

SubsetSum $\in$ NP-complete

- Size of the created instance:
  - $|A| = 2n + 2m$
  - each created integer has at most $n + m$ digits (including t)
    $\rightarrow$ integers in the interval $[0, 10^{n+m}]$
    $\rightarrow$ binary representation: at most $\log_2 10^{n+m} = O(n + m)$ bits

On the example $(x_1 \vee \bar{x}_2 \vee x_3) \wedge (x_2 \vee x_3 \vee x_4) \wedge (\bar{x}_1 \vee \bar{x}_3 \vee x_4)$
a feasible assigment is: $x_1 = x_4 = T$, $x_2 = x_3 = F$

- B contains:
  $1000100, 100100, 10001, 1011$
  $200, 10, 20, 2$
- $t = 1111444$

SubsetSum $\in$ NP-complete

- Size of the created instance:
  - $|A| = 2n + 2m$
  - each created integer has at most $n + m$ digits (including t)
    $\rightarrow$ integers in the interval $[0, 10^{n+m}]$
    $\rightarrow$ binary representation: at most $\log_2 10^{n+m} = O(n+m)$ bits

On the example $(x_1 \vee \bar{x}_2 \vee x_3) \wedge (x_2 \vee x_3 \vee x_4) \wedge (\bar{x}_1 \vee \bar{x}_3 \vee x_4)$
a feasible assigment is: $x_1 = x_4 = T$, $x_2 = x_3 = F$

- B contains:
  $1000100, 100100, 10001, 1011$
  $200, 10, 20, 2$
- $t = 1111444$

- $\mathcal{F}$ is satisfiable iff there is a set $B \subseteq A$ with $\sum_{a_i \in B} a_i = t$

$(\Rightarrow)$

- assume that $\mathcal{F}$ is satisfiable

($\Rightarrow$)

- assume that $\mathcal{F}$ is satisfiable
- for each x_i:
  – if $x_i = \mathrm{TRUE}$, then add y_i to B
  – if $x_i = \mathrm{FALSE}$, then add z_i to B
- for each C_j:
  – if 1 literal is TRUE, then add both g_j and h_j in B
  – if 2 literals are TRUE, then add h_j in B
  – if 3 literals are TRUE, then add g_j in B

($\Rightarrow$)

- assume that $\mathcal{F}$ is satisfiable

- for each x_i:
  – if $x_i = \text{TRUE}$, then add y_i to B
  – if $x_i = \text{FALSE}$, then add z_i to B
- for each C_j:
  – if 1 literal is TRUE, then add both g_j and h_j in B
  – if 2 literals are TRUE, then add h_j in B
  – if 3 literals are TRUE, then add g_j in B

- B is a $\textsc{SubsetSum}$
  – left part of t: we select only one of y_i and z_i, for each $1 \leq i \leq n$
  – right part of t: we select g_j and h_j in order to have exactly 4 for each clause

SubsetSum $\in$ NP-complete

3. $\mathcal{F}$ is satisfiable iff there is a set $B \subseteq A$ with $\sum_{a_i \in B} a_i = t$

   $(\Leftarrow)$

   - assume there is a set B such that $\sum_{a_i \in B} a_i = t$
   - each column contains at most 6 ones (3 at the top and 3 at the bottom)

SubsetSum $\in$ NP-complete

3. $\mathcal{F}$ is satisfiable iff there is a set $B \subseteq A$ with $\sum_{a_i \in B} a_i = t$

   $(\Leftarrow)$

   - assume there is a set B such that $\sum_{a_i \in B} a_i = t$
   - each column contains at most 6 ones (3 at the top and 3 at the bottom)
   - there is no other way to have 1 in the variable-left part of t except from selecting exactly one of each y_i and z_i

SubsetSum $\in$ NP-complete

3. $\mathcal{F}$ is satisfiable iff there is a set $B \subseteq A$ with $\sum_{a_i \in B} a_i = t$

   $(\Leftarrow)$

   - assume there is a set B such that $\sum_{a_i \in B} a_i = t$
   - each column contains at most 6 ones (3 at the top and 3 at the bottom)
   - there is no other way to have 1 in the variable-left part of t except from selecting exactly one of each y_i and z_i
   - then, set:
     – $x_i = \text{TRUE}$, if $y_i \in B$
     – $x_i = \text{FALSE}$, if $z_i \in B$

SubsetSum $\in$ NP-complete

3. $\mathcal{F}$ is satisfiable iff there is a set $B \subseteq A$ with $\sum_{a_i \in B} a_i = t$
   $(\Leftarrow)$
   - assume there is a set B such that $\sum_{a_i \in B} a_i = t$
   - each column contains at most 6 ones (3 at the top and 3 at the bottom)
   - there is no other way to have 1 in the variable-left part of t except from selecting exactly one of each y_i and z_i
   - then, set:
     – $x_i = \text{TRUE}$, if $y_i \in B$
     – $x_i = \text{FALSE}$, if $z_i \in B$
   - there is no way to have 4 in the clause-right part of t by selecting only g_j and h_j

SubsetSum $\in$ NP-complete

3. $\mathcal{F}$ is satisfiable iff there is a set $B \subseteq A$ with $\sum_{a_i \in B} a_i = t$

   $(\Leftarrow)$

   - assume there is a set B such that $\sum_{a_i \in B} a_i = t$
   - each column contains at most 6 ones (3 at the top and 3 at the bottom)
   - there is no other way to have 1 in the variable-left part of t except from selecting exactly one of each y_i and z_i
   - then, set:
     – $x_i = \text{TRUE}$, if $y_i \in B$
     – $x_i = \text{FALSE}$, if $z_i \in B$
   - there is no way to have 4 in the clause-right part of t by selecting only g_j and h_j
   - thus, at least one literal (y_i, z_i) should be one for each clause column
   - therefore, this assignment satisfies $\mathcal{F}$

An algorithm for SubsetSum

- Dynamic Programming

An algorithm for SubsetSum

- Dynamic Programming
- consider the integers sorted in non-decreasing order: $a_1 \leq a_2 \leq \ldots \leq a_n$
- $S[i,q] = \begin{cases} \text{True,} & \text{if there is a SubsetSum among the } i \text{ first} \\ & \text{integers which sums up exactly to } q \\ \text{False,} & \text{otherwise} \end{cases}$

An algorithm for SubsetSum

- Dynamic Programming
- consider the integers sorted in non-decreasing order: $a_1 \leq a_2 \leq \ldots \leq a_n$
- $S[i, q] = \begin{cases} \text{True}, & \text{if there is a SubsetSum among the } i \text{ first} \\ & \text{integers which sums up exactly to } q \\ \text{False}, & \text{otherwise} \end{cases}$

Algorithm

1: Initialization:
   - $S[i, 0] = \text{True}$, for any $i \geq 1$
   - $S[1, q] = \begin{cases} \text{True}, & \text{if } q = a_1 \\ \text{False}, & \text{otherwise} \end{cases}$

An algorithm for SUBSETSUM

- Dynamic Programming
- consider the integers sorted in non-decreasing order: $a_1 \leq a_2 \leq \ldots \leq a_n$
- $S[i,q] = \begin{cases} \text{True}, & \text{if there is a SubsetSum among the } i \text{ first} \\ & \text{integers which sums up exactly to } q \\ \text{False}, & \text{otherwise} \end{cases}$

Algorithm

```
Initialization:
  – S[i,0] = True, for any i ≥ 1
  – S[1,q] = { True,  if q = a_1
             { False, otherwise
for i = 1 to n do
    for q = 1 to t do
        S[i,q] = S[i−1,q] or S[i−1,q−a_i]
```

An algorithm for SubsetSum

- Example: $A = \{2, 3, 4, 6, 8\}$ and $t = 11$

An algorithm for SUBSETSUM

- Example: $A = \{2, 3, 4, 6, 8\}$ and $t = 11$

		q											
		0	1	2	3	4	5	6	7	8	9	10	11
	1	T											
	2	T											
i	3	T											
	4	T											
	5	T											

$S[i, 0] = \text{True}$, for any $i \geq 1$

An algorithm for SubsetSum

- Example: $A = \{2, 3, 4, 6, 8\}$ and $t = 11$

		q											
		0	1	2	3	4	5	6	7	8	9	10	11
	1	T	F	T	F	F	F	F	F	F	F	F	F
	2	T											
i	3	T											
	4	T											
	5	T											

$$S[1, q] = \begin{cases} \text{True}, & \text{if } q = a_1 \\ \text{False}, & \text{otherwise} \end{cases}$$

An algorithm for SubsetSum

- Example: $A = \{2, 3, 4, 6, 8\}$ and $t = 11$

		q											
		0	1	2	3	4	5	6	7	8	9	10	11
	1	T	F	T	F	F	F	F	F	F	F	F	F
	2	T	F	T									
i	3	T											
	4	T											
	5	T											

$S[i, q] = S[i-1, q]$ **or** $S[i-1, q-a_i]$
$S[2, 2] = S[1, 2]$ **or** $S[1, -1]$

An algorithm for SubsetSum

- Example: $A = \{2, 3, 4, 6, 8\}$ and $t = 11$

		q											
		0	1	2	3	4	5	6	7	8	9	10	11
	1	T	F	T	F	F	F	F	F	F	F	F	F
	2	T	F	T	T								
i	3	T											
	4	T											
	5	T											

$S[i, q] = S[i-1, q]$ **or** $S[i-1, q-a_i]$
$S[2, 3] = S[1, 3]$ **or** $S[1, 0]$

An algorithm for SubsetSum

- Example: $A = \{2, 3, 4, 6, 8\}$ and $t = 11$

		q											
		0	1	2	3	4	5	6	7	8	9	10	11
	1	T	F	T	F	F	F	F	F	F	F	F	F
	2	T	F	T	T	F	T						
i	3	T											
	4	T											
	5	T											

$S[i, q] = S[i-1, q]$ **or** $S[i-1, q-a_i]$
$S[2, 5] = S[1, 5]$ **or** $S[1, 2]$

An algorithm for SUBSETSUM

- Example: $A = \{2, 3, 4, 6, 8\}$ and $t = 11$

		q											
		0	1	2	3	4	5	6	7	8	9	10	11
	1	T	F	T	F	F	F	F	F	F	F	F	F
	2	T	F	T	T	F	T	F	F	F	F	F	F
i	3	T											
	4	T											
	5	T											

$S[i, q] = S[i-1, q]$ **or** $S[i-1, q-a_i]$

An algorithm for SUBSETSUM

- Example: $A = \{2, 3, 4, 6, 8\}$ and $t = 11$

		q											
		0	1	2	3	4	5	6	7	8	9	10	11
	1	T	F	T	F	F	F	F	F	F	F	F	F
	2	T	F	T	T	F	T	F	F	F	F	F	F
i	3	T	F	T	T	T							
	4	T											
	5	T											

$S[i, q] = S[i-1, q]$ **or** $S[i-1, q-a_i]$
$S[3, 4] = S[2, 4]$ **or** $S[2, 0]$

An algorithm for SUBSETSUM

- Example: $A = \{2, 3, 4, 6, 8\}$ and $t = 11$

		q											
		0	1	2	3	4	5	6	7	8	9	10	11
	1	T	F	T	F	F	F	F	F	F	F	F	F
	2	T	F	T	T	F	T	F	F	F	F	F	F
i	3	T	F	T	T	T	T	T					
	4	T											
	5	T											

$S[i, q] = S[i-1, q]$ **or** $S[i-1, q-a_i]$
$S[3, 6] = S[2, 6]$ **or** $S[2, 2]$

An algorithm for SubsetSum

- Example: $A = \{2, 3, 4, 6, 8\}$ and $t = 11$

		0	1	2	3	4	5	6	7	8	9	10	11
	1	T	F	T	F	F	F	F	F	F	F	F	F
	2	T	F	T	T	F	T	F	F	F	F	F	F
i	3	T	F	T	T	T	T	T	T	F	T	F	F
	4	T											
	5	T											

$S[i, q] = S[i-1, q]$ **or** $S[i-1, q-a_i]$

An algorithm for SubsetSum

- Example: $A = \{2, 3, 4, 6, 8\}$ and $t = 11$

		q											
		0	1	2	3	4	5	6	7	8	9	10	11
	1	T	F	T	F	F	F	F	F	F	F	F	F
	2	T	F	T	T	F	T	F	F	F	F	F	F
i	3	T	F	T	T	T	T	T	T	F	T	F	F
	4	T	F	T	T	T	T	T	T	T	T	T	T
	5	T											

$S[i, q] = S[i-1, q]$ **or** $S[i-1, q-a_i]$

An algorithm for SubsetSum

- Example: $A = \{2, 3, 4, 6, 8\}$ and $t = 11$

		q											
		0	1	2	3	4	5	6	7	8	9	10	11
	1	T	F	T	F	F	F	F	F	F	F	F	F
	2	T	F	T	T	F	T	F	F	F	F	F	F
i	3	T	F	T	T	T	T	T	T	F	T	F	F
	4	T	F	T	T	T	T	T	T	T	T	T	T
	5	T	F	T	T	T	T	T	T	T	T	T	T

- there is a TRUE in column $q = 11$, hence $\langle A, t \rangle \in$ SubsetSum

An algorithm for SubsetSum

- Example: $A = \{2, 3, 4, 6, 8\}$ and $t = 11$

		q											
		0	1	2	3	4	5	6	7	8	9	10	11
	1	T	F	T	F	F	F	F	F	F	F	F	F
	2	T	F	T	T	F	T	F	F	F	F	F	F
i	3	T	F	T	T	T	T	T	T	F	T	F	F
	4	T	F	T	T	T	T	T	T	T	T	T	T
	5	T	F	T	T	T	T	T	T	T	T	T	T

- how to construct the set B ?

An algorithm for SubsetSum

- Example: $A = \{2, 3, 4, 6, 8\}$ and $t = 11$

		q											
		0	1	2	3	4	5	6	7	8	9	10	11
	1	T	F	T	F	F	F	F	F	F	F	F	F
	2	T	F	T	T	F	T	F	F	F	F	F	F
i	3	T	F	T	T	T	T	T	T	F	T	F	F
	4	T	F	T	T	T	T	T	T	T	T	T	T
	5	T	F	T	T	T	T	T	T	T	T	T	T

- how to construct the set B ?
- $S[5, 11] = S[4, 11]$ **or** $S[4, 3]$
  - $S[4, 11]$: $a_5 \notin B$
  - $S[4, 3]$: $a_5 \in B$

An algorithm for SUBSETSUM

- Example: $A = \{2, 3, 4, 6, 8\}$ and $t = 11$

		q											
		0	1	2	3	4	5	6	7	8	9	10	11
	1	T	F	T	F	F	F	F	F	F	F	F	F
	2	T	F	T	T	F	T	F	F	F	F	F	F
i	3	T	F	T	T	T	T	T	T	F	T	F	F
	4	T	F	T	T	T	T	T	T	T	T	T	T
	5	T	F	T	T	T	T	T	T	T	T	T	T

- how to construct the set B ?
- $S[5, 11] = S[4, 11]$ **or** $S[4, 3]$
  - $S[4, 11]$: $a_5 \notin B$
  - $S[4, 3]$: $a_5 \in B$
- $S[4, 3] = S[3, 3]$ **or** $S[3, -3]$, so $a_4 \notin B$

An algorithm for SUBSETSUM

- Example: $A = \{2, 3, 4, 6, 8\}$ and $t = 11$

		q											
		0	1	2	3	4	5	6	7	8	9	10	11
	1	T	F	T	F	F	F	F	F	F	F	F	F
	2	T	F	T	T	F	T	F	F	F	F	F	F
i	3	T	F	T	T	T	T	T	T	F	T	F	F
	4	T	F	T	T	T	T	T	T	T	T	T	T
	5	T	F	T	T	T	T	T	T	T	T	T	T

- how to construct the set B ?
- $S[5, 11] = S[4, 11]$ **or** $S[4, 3]$
  - $S[4, 11]$: $a_5 \notin B$
  - $S[4, 3]$: $a_5 \in B$
- $S[4, 3] = S[3, 3]$ **or** $S[3, -3]$, so $a_4 \notin B$
- $S[3, 3] = S[2, 3]$ **or** $S[2, -1]$, so $a_3 \notin B$

An algorithm for SubsetSum

- Example: $A = \{2, 3, 4, 6, 8\}$ and $t = 11$

		q											
		0	1	2	3	4	5	6	7	8	9	10	11
	1	T	F	T	F	F	F	F	F	F	F	F	F
	2	T	F	T	T	F	T	F	F	F	F	F	F
i	3	T	F	T	T	T	T	T	T	F	T	F	F
	4	T	F	T	T	T	T	T	T	T	T	T	T
	5	T	F	T	T	T	T	T	T	T	T	T	T

- how to construct the set B ?
- $S[5, 11] = S[4, 11]$ **or** $S[4, 3]$
  - $S[4, 11]$: $a_5 \notin B$
  - $S[4, 3]$: $a_5 \in B$
- $S[4, 3] = S[3, 3]$ **or** $S[3, -3]$, so $a_4 \notin B$
- $S[3, 3] = S[2, 3]$ **or** $S[2, -1]$, so $a_3 \notin B$
- $S[2, 3] = S[1, 3]$ **or** $S[1, 0]$, so $a_2 \in B$, $a_1 \notin B$

Dynamic Programming for SubsetSum

- Complexity in $O(n \cdot t)$

Dynamic Programming for SubsetSum

- Complexity in $O(n \cdot t)$
- Is this polynomial?

Dynamic Programming for SubsetSum

- Complexity in $O(n \cdot t)$
- Is this polynomial?
- **NO!** if yes, then $\mathrm{P} = \mathrm{NP}$

- input: $I = \langle A, t \rangle$
- size of the input:

$$|I| = \log_2 t + \sum_{a_i \in A} \log_2 a_i$$

Dynamic Programming for SubsetSum

- Complexity in $O(n \cdot t)$
- Is this polynomial?
- **NO!** if yes, then $\mathrm{P} = \mathrm{NP}$

- input: $I = \langle A, t \rangle$
- size of the input:

$$|I| = \log_2 t + \sum_{a_i \in A} \log_2 a_i = O(\log_2 t)$$

Dynamic Programming for SubsetSum

- Complexity in $O(n \cdot t)$
- Is this polynomial?
- **NO!** if yes, then $\mathrm{P} = \mathrm{NP}$

- input: $I = \langle A, t \rangle$
- size of the input:

$$|I| = \log_2 t + \sum_{a_i \in A} \log_2 a_i = {\color{red}O(\log_2 t)}$$

- complexity of the algorithm:

$$O(n \cdot t) = O(n \cdot 2^{|I|})$$

Dynamic Programming for SubsetSum

- Complexity in $O(n \cdot t)$
- Is this polynomial?
- **NO!** if yes, then $\mathrm{P} = \mathrm{NP}$

- input: $I = \langle A, t \rangle$
- size of the input:

$$|I| = \log_2 t + \sum_{a_i \in A} \log_2 a_i = O(\log_2 t)$$

- complexity of the algorithm:

$$O(n \cdot t) = O(n \cdot 2^{|I|})$$

- that is, exponential to the size of the input !

Pseudo-polynomial algorithms

- $|I|_1$: the encoding of the input in unary

Pseudo-polynomial algorithms

- $|I|_1$: the encoding of the input in unary
- example: $\textsc{SubsetSum}$

$$|I|_1 = t + \sum_{a_i \in A} a_i$$

Pseudo-polynomial algorithms

- $|I|_1$: the encoding of the input in unary
- example: SubsetSum

$$|I|_1 = t + \sum_{a_i \in A} a_i$$

then the complexity of the algorithm is polynomial:

$$O(n \cdot t) = O(n \cdot |I|_1)$$

Pseudo-polynomial algorithms

- $|I|_1$: the encoding of the input in unary
- example: SubsetSum

$$|I|_1 = t + \sum_{a_i \in A} a_i$$

  then the complexity of the algorithm is polynomial:

$$O(n \cdot t) = O(n \cdot |I|_1)$$

- Definition: we call an algorithm **pseudo-polynomial** if its complexity is polynomial to the size of the input, when this is encoded in unary.

Pseudo-polynomial algorithms

- $|I|_1$: the encoding of the input in unary
- example: SubsetSum

$$|I|_1 = t + \sum_{a_i \in A} a_i$$

then the complexity of the algorithm is polynomial:

$$O(n \cdot t) = O(n \cdot |I|_1)$$

- Definition: we call an algorithm **pseudo-polynomial** if its complexity is polynomial to the size of the input, when this is encoded in unary.
- Definition: NP-complete problems that admit a pseudo-polynomial algorithm are called **weakly** NP-complete.

Pseudo-polynomial algorithms

- $|I|_1$: the encoding of the input in unary
- example: SubsetSum

$$|I|_1 = t + \sum_{a_i \in A} a_i$$

  then the complexity of the algorithm is polynomial:

$$O(n \cdot t) = O(n \cdot |I|_1)$$

- Definition: we call an algorithm **pseudo-polynomial** if its complexity is polynomial to the size of the input, when this is encoded in unary.
- Definition: NP-complete problems that admit a pseudo-polynomial algorithm are called **weakly** NP-complete.
- Definition: we call a problem **strong** or **unary** NP-complete if it remains NP-complete even when the input is encoded in unary.

Observations

- where is the problem with the reduction of SubsetSum if the input is encoded in unary?

Observations

- where is the problem with the reduction of SubsetSum if the input is encoded in unary?
  - each created integer has at most $n + m$ digits (including t)
    $\rightarrow$ integers in the interval $[0, 10^{n+m}]$
    $\rightarrow$ unary representation: 10^{n+m} symbols per integer
  - the size of the created input is **not** polynomial with respect to the size of the initial input

Observations

- where is the problem with the reduction of SubsetSum if the input is encoded in unary?
  - each created integer has at most $n + m$ digits (including t)
    $\rightarrow$ integers in the interval $[0, 10^{n+m}]$
    $\rightarrow$ unary representation: 10^{n+m} symbols per integer
  - the size of the created input is **not** polynomial with respect to the size of the initial input
- are there numerical problems that are strongly NP-complete?

Observations

- where is the problem with the reduction of SubsetSum if the input is encoded in unary?
  - each created integer has at most $n + m$ digits (including t)
    $\rightarrow$ integers in the interval $[0, 10^{n+m}]$
    $\rightarrow$ unary representation: 10^{n+m} symbols per integer
  - the size of the created input is **not** polynomial with respect to the size of the initial input
- are there numerical problems that are strongly NP-complete?
  - YES
  - 3-Partition, Bin-Packing, ...

Observations

- where is the problem with the reduction of SUBSETSUM if the input is encoded in unary?
  - each created integer has at most $n + m$ digits (including t)
    $\rightarrow$ integers in the interval $[0, 10^{n+m}]$
    $\rightarrow$ unary representation: 10^{n+m} symbols per integer
  - the size of the created input is **not** polynomial with respect to the size of the initial input
- are there numerical problems that are strongly NP-COMPLETE?
  - YES
  - 3-PARTITION, BIN-PACKING, ...
- Attention! if $A \leq_{\mathrm{P}} B$ and A is weakly NP-COMPLETE, then we only prove that B is weakly NP-COMPLETE

Exercise: 3-Partition problem

3-Partition

Input: a set of positive integers $S = \{s_1, s_2, \ldots, s_{3n}\}$,
where $\sum_{s_i \in S} = n \cdot t$ and $\frac{t}{4} \leq s_i \leq \frac{t}{2}$ for each $s_i \in A$

Question: can S be partitioned into n disjoint sets $S_1, S_2, \ldots, S_n$
such that $\sum_{s_i \in S_j} s_i = t$, for $1 \leq j \leq n$?

Exercise: 3-Partition problem

3-Partition

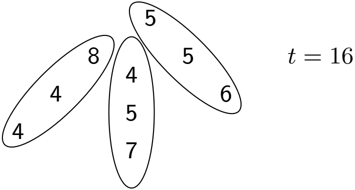
Input: a set of positive integers $S = \{s_1, s_2, \ldots, s_{3n}\}$,
where $\sum_{s_i \in S} = n \cdot t$ and $\frac{t}{4} \leq s_i \leq \frac{t}{2}$ for each $s_i \in A$

Question: can S be partitioned into n disjoint sets $S_1, S_2, \ldots, S_n$
such that $\sum_{s_i \in S_j} s_i = t$, for $1 \leq j \leq n$?

Example

$S = \{4, 4, 4, 5, 5, 5, 6, 7, 8\}$ with the target: $t = 16$

- observation: each S_j should have exactly 3 integers.
  Here is a solution:



- Show that 3-PARTITION is NP-COMPLETE in the strong sense

NP-hardness

A problem A is NP-HARD if any problem $B \in \mathcal{NP}$ is polynomially time reducible to A.

NP-HARDNESS

A problem A is NP-HARD if any problem $B \in \mathcal{NP}$ is polynomially time reducible to A.

- A is not necessarily in $\mathcal{NP}$!
- A is not necessarily a *decision* problem
- it is enough to show that there is a $B \in \mathcal{NP}$ such that $B \leq_{\mathcal{P}} A$

NP-HARDNESS

A problem A is NP-HARD if any problem $B \in \mathcal{NP}$ is polynomially time reducible to A.

- A is not necessarily in $\mathcal{NP}$!
- A is not necessarily a *decision* problem
- it is enough to show that there is a $B \in \mathcal{NP}$ such that $B \leq_{\mathcal{P}} A$