NP-Completeness and Classes P and NP

The class P: consists of those problems that are solvable in polynomial time. These problems that can be solved in time $O(n^k)$ for some constant k, where n is the size of input to the problem.

- The class NP:- consists of those problems that are verifiable in polynomial time.
- The class NPC (NP-complete):- if it is in NP and is as hard as any problem in NP.

Optimization problems:- in which each feasible solution has an associated value and we wish to find feasible solution with the best value.
eg. in shortest-path problem, we are given a graph G and vertices u and v, and we wish to find path from u to v that uses fewest edges.

Decision problems:- in which answer is simply "yes" or "no".

Note: NP-completeness applies directly not to optimization problems, but to decision problems.

checking whether to = E, S related

- The problems solved by polynomial-time algorithms are tractable or easy.

Superpolynomial time algorithms : they require superpolynomial running time (eg. $O(n!)$).
~~These algos are intractable or hard.~~

Abstract problem :- Q is a binary relation on a set I of problem instances and a set S of problem solutions.

eg. In shortest path problem, instance consists of a graph and 2 vertices, the solution is sequence of vertices in a graph with empty sequence denoting that no path exists

Encodings
An encoding of a set S of objects is a mapping from S to a set of binary strings. eg. encoding of natural numbers $N = \{0, 1, 2, 3, 4 - -\}$ is strings $\{0, 1, 10, 11, 100 - -\}$.

Concrete problem: A problem whose instance set is set of binary strings.

...inal-language framework

Alphabet: Σ is a finite set of symbols.

Language: L over Σ is any set of strings made up of symbols from Σ.

eg. if $\Sigma = \{0, 1\}$, set $L = \{10, 11, 101, 111, 1011, \ldots\}$ (5)

- Empty string is denoted by ε.
- Empty language is denoted by ϕ.
- Language of all strings over Σ is denoted by Σ^*.

eg. if $\Sigma = \{0, 1\}$, then $\Sigma^* = \{\varepsilon, 0, 1, 00, 01, 10, 11 \ldots\}$ is set of all binary strings.

- We can perform union and intersection on languages.
- Complement of L, is denoted by $\bar{L} = \Sigma^* - L$
- Concatenation of 2 languages L_1 and L_2 is the language:

$$L = \{x_1 x_2 : x_1 \in L_1 \text{ and } x_2 \in L_2\}.$$

- Closure or kleene star of a language L is language:-

$$L^* = \{\varepsilon\} \cup L \cup L^2 \cup L^3 \cup \ldots$$

where L^k is language obtained by concatenating L to itself k times.

...cking whether $t = \Sigma_{s \in S}$

- The formal-language framework allows to express the relation between decision problems and algorithms that solve them.

We say that an algo A accepts a string $x \in \{0,1\}^*$ if given input x, the algo's output A(x) is 1.

- An algo A rejects a string x if A(x) = 0.

Polynomial-time verification

- We examine a problem for which we know of no polynomial-time decision algo yet, verification is easy.

Hamiltonian cycles

- A hamiltonian cycle of an undirected graph G = (V, E) is a simple cycle that contains each vertex in V.

- A graph that contains a hamiltonian cycle is said to be hamiltonian, otherwise it is nonhamiltonian.

- Given a problem instance (G), one possible decision algo lists all permutations of the vertices of G and then checks each permutation to see if it is a hamiltonian path.

running time of the algorithm for n vertices graph is $= \Omega(n!)$

This algo does not run in polynomial time.

Verification algorithms

- Suppose that it is given that graph G is hamiltonian and the vertices in order along hamiltonian cycle is given. It would be easy to verify that provided cycle is hamiltonian by checking whether it is permutation of vertices of V and whether each of consecutive edges along cycle actually exists in graph.
- This verification algo can be implemented to run in $O(n^2)$ time where n is length of encoding of G.
- Thus proof that a hamiltonian cycle exists in a graph can be verified in polynomial time.
- A verification algorithm:- is a two-argument algo A where one argument ~~is~~ is an ordinary input string x and other is a binary string y called a certificate. A two-argument algo A verifies an input string x if there exits a certificate y such that $A(x, y) = 1$.

- The language verified by a verification algo

$$L = \{x \in \{0,1\}^* : \text{there exists } y \in \{0,1\}^* \text{ such that } A(x,y) = 1\}.$$

Complexity class NP

The complexity class NP is class of language that can be verified by a polynomial-time algo.

- A language L belongs to NP if and only if there exist a two input polynomial-time algo A and constant c such that;

$$L = \{x \in \{0,1\}^* : \text{there exists a certificate } y \text{ with } |y| = O(|x|^c) \text{ such that } A(x,y) = 1\}$$

- We say that algo A verifies language L in polynomial time.

NP-completeness and reducibility

NPC class has a surprising property that if any NP-complete problem can be solved in polynomial time, then every problem in NP has a polynomial-time solution. But, no polynomial time algo has ever been discovered for any NP-complete problem.

ibility

problem Q can be reduced to another problem Q' if any instance of Q can be easily rephrased as an instance of Q', the solution to which provides a solution to instance of Q.

eg. problem of solving linear equations reduces to problem of solving quadratic equation. Given an instance $ax+b=0$, we transform it to $0x^2+ax+b=0$, whose solution provides a solution to $ax+b=0$

- We say that a language L_1 is polynomial time reducible to a language L_2, written $L_1 \leq_p L_2$, if there exists a polynomial-time computable function $f: \{0,1\}^* \rightarrow \{0,1\}^*$ such that for all $x \in \{0,1\}^*$,

$x \in L_1$ if and only if $f(x) \in L_2$.

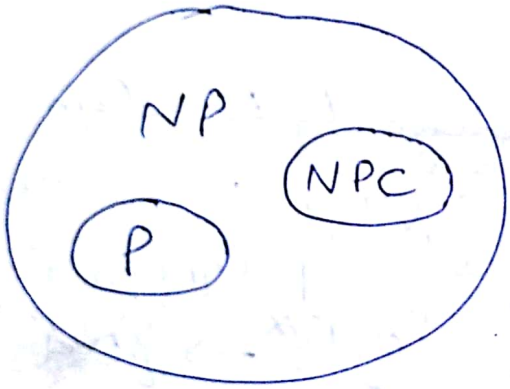
We call function f the reduction function, and a polynomial-time algorithm F that computes f is called reduction algorithm.

NP- completeness

A language $L \subseteq \{0,1\}^*$ is NP-complete if

1. $L \in NP$ and
2. $L' \leq_P L$ for every $L' \in NP$

If a language L satisfies property 2, but not necessarily property 1, we say that L is NP-hard.



This is how most theoretical computer scientists view the relationship among P, NP and NPC. Both P + NPC are wholly contained within NP and $P \cap NPC = \emptyset$.

Scanned by CamScanner

ADA Notes
CSE-1, 4th sem

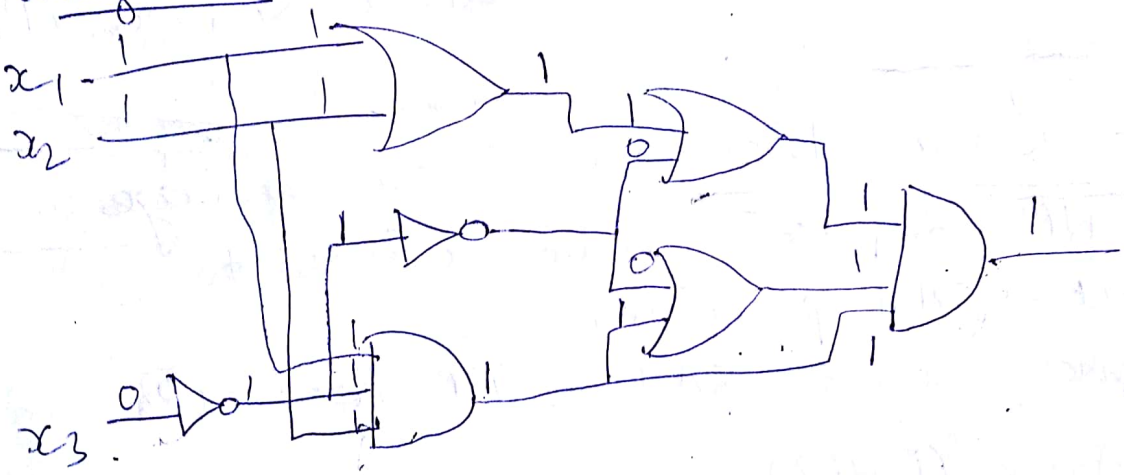
Tasleen Kaur

(1)

NP-completeness proofs

(1) Circuit satisfiability problem

This problem takes as input a boolean circuit with a single output node and asks whether there is an assignment of values to the circuit's inputs so that its output value is "1". Such an assignment of values is called satisfying assignment



This circuit is satisfiable as assignment of values $(x_1=1, x_2=1, x_3=0)$ to inputs of the circuit causes output to be 1.

Circuit-SAT problem belongs to class

Proof: If given circuit is satisfiable, then algorithm guess the values of input nodes as well as output values of each logic gate in circuit. Then the algo simply visits each logic gate in the circuit. For each logic gate, it checks whether the guessed output for guessed inputs to the gate is correct or not. Using this procedure, we check e (output of circuit) in polynomial time. If e is 1, the algorithm will output "Yes". ∴ Circuit-SAT ∈ NP — (1)

Cook-Levin Theorem: Every language in class NP-complete can be reduced to Circuit-SAT problem.
Therefore Circuit-SAT is NP-hard — (2)

∴ From (1) + (2)
Circuit-SAT is NP-complete.

---

(2) Satisfiability problem

Given an expression e, is there an assignment of values to the variables $x_1, x_2, \ldots x_n$ that will make the value of e true?

Eg. $e = (\bar{x}_1 \vee x_2) \wedge (x_1 \vee x_3)$
If $x_1 = 0$, $x_2 = 1$, $x_3 = 1$, then e is true.
∴ e is satisfiable.

to solve SAT problem is to (2)
all possible values for the variables
$x_1, x_2 -- x_n$ and evaluate for e.
In this case $T(n) = O(2^n)$
$\therefore$ SAT $\notin$ P

If e is satisfiable, we can verify it by guessing the value of each x_i and then evaluate e.
In this case, $T(n) = O(n)$
$\therefore$ SAT $\in$ NP — (1)

SAT problem is NP-complete

Proof: we have proved that SAT $\in$ NP (from (1))

Now we will show that we can reduce circuit-SAT problem to SAT problem.
For each circuit, we can find the equivalent boolean formula
$\therefore$ SAT is NP-hard — (2)
From (1) + (2), SAT is NP-complete.

(3) 3-CNF satisfiability

This problem takes a boolean formula s that is in CNF (conjunctive normal form) with each elementary sum in s having exactly three literals and asks if s is satisfiable.

eg. $s = (\bar{x_1} + x_2 + \bar{x_7})(x_3 + \bar{x_5} + x_6)$
$(x_2 + x_4 + \bar{x_6})(x_1 + x_5 + x_8)$

As we can design nondeterministic algo to verify this problem in polynomial time

$\therefore$ 3-SAT $\in$ NP. — (1)

We can reduce SAT to 3 CNF-SAT as follows:-

For each clause C_i in C,

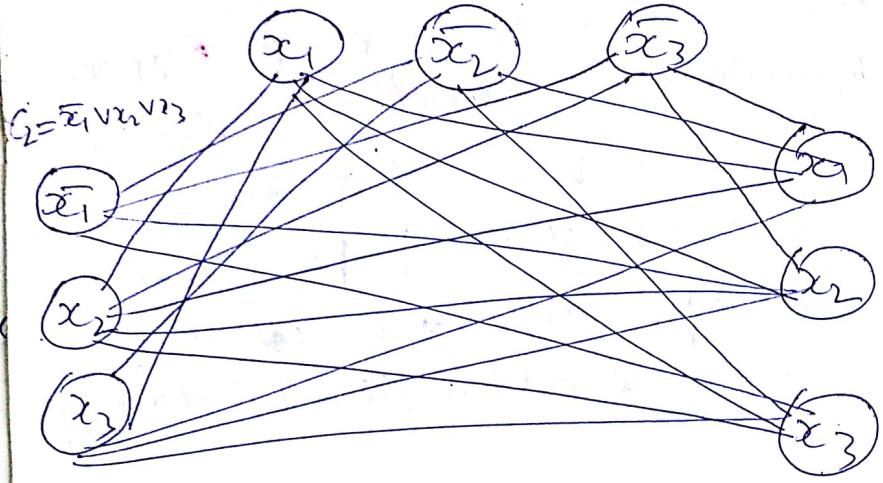
(1) If $C_i = (a)$, ie it has one literals then we can write $C_i = (a+b+c)(a+\bar{b}+c)(a+b+\bar{c})(a+\bar{b}+\bar{c})$ where b and c are new variables not used anywhere else.

(2) If $C_i = (a+b)$, then we can write $C_i = (a+b+c)(a+b+\bar{c})$ where c is new variable not used anywhere else.

...t an edge between 2 vertices v_i^r and v_j^s if both of following holds:-
- v_i^r and v_j^s are in different triples ie $r \neq s$
- their corresponding literals are consistent ie l_i^r is not negation of l_j^s.

eg. $\phi = (x_1 \vee \bar{x}_2 \vee \bar{x}_3) \wedge (\bar{x}_1 \vee x_2 \vee x_3) \wedge (x_1 \vee x_2 \vee x_3)$

$C_1 = x_1 \vee \bar{x}_2 \vee \bar{x}_3$



$C_3 = x_1 \vee x_2 \vee x_3$

for ϕ, assignment is $x_1=0, x_2=0, x_3=1$,
A corresponding clique of size $k=3$ consists of vertices $(\bar{x}_2, x_3, x_3)$

...let, for an ... whether ...

Vertex-cover problem

A vertex-cover of an undirected graph $G=(V,E)$ is a subset $V' \subseteq V$ such that if $(u,v) \in E$, then $v \in V'$ or $u \in V'$ (or both). ie each vertex covers its incident edges and a vertex cover for G is a set of vertices that covers all edges in E.
Size of vertex cover is no. of vertices in it.
Vertex-cover problem is to find vertex cover of minimum size in a given graph.

Vertex-cover problem is NP-complete

We first show that vertex-cover $\in$ NP. Suppose we are given a graph $G=(V,E)$ and an integer k. As a certificate we choose $V' \subseteq V$ is vertex cover. The verification algo checks that $|V'|=k$ and then checks for each edge $(u,v) \in E$ whether $u \in V'$ or $v \in V'$. This verification can be done in polynomial time.

We reduce clique to vertex cover problem. This reduction is based on notion of complement of a graph. Given an undirected graph $G=(V,E)$, we define complement of G as $\bar{G}=(V,\bar{E})$, where $\bar{E} = \{(u,v) : (u,v) \notin E\}$ ie $\bar{G}$ is graph containing those edges that are not in G.

~~clique is in G and vertex cover is~~

$(a_1+a_2$ can write
$= (a_1+a_2 -$
$+ - - - (b_{k-3}+$
where b_1, b_2

$= (a_1 + a_2 + \cdots a_k)$ where $k > 3$

we can write

$C_i = (a_1 + a_2 + b_1)(\bar{b_1} + a_3 + b_2)(\bar{b_2} + a_4 + b_3)$
$+ \cdots (\bar{b_{k-3}} + a_{k-1} + a_k)$

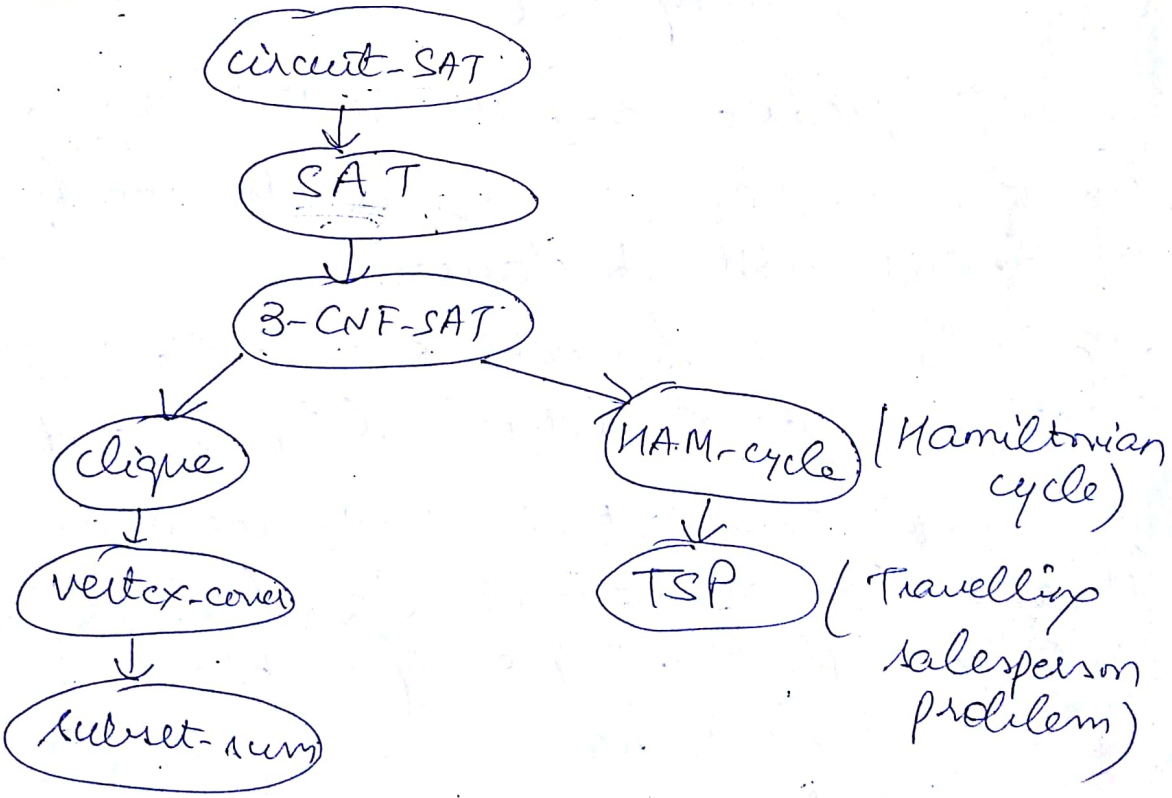
where $b_1, b_2 \cdots b_{k-3}$ are new variables not used anywhere else.

So, we can reduce SAT to 3-SAT problem in polynomial time.

$\therefore$ 3-CNF SAT problem is NP-hard — (2)

From (1) & (2), 3 CNF-SAT problem is NP-complete.

NP-complete problems



Clique problem

A clique in an undirected graph $G=(V,E)$ is a subset $V' \subseteq V$ of vertices, each pair of which is connected by an edge in E. A clique is a complete subgraph of G. The size of clique is number of vertices it contains. Clique problem is optimization problem of finding a clique of maximum size in a graph.

The clique problem is NP-complete

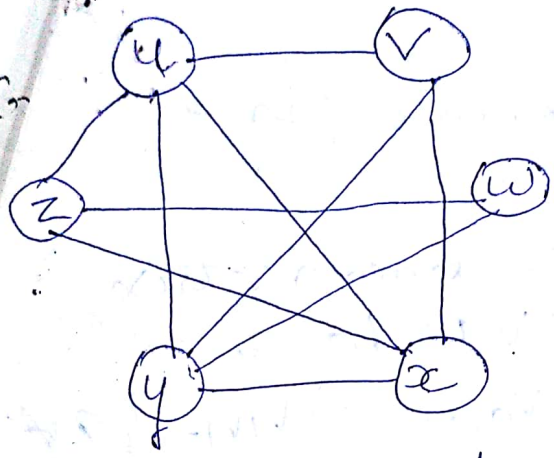
Proof: To show that clique $\in NP$, for a given graph $G=(V,E)$, we use set $V' \subseteq V$ of vertices in clique as certificate for G. Checking whether V' is a clique can be done in polynomial time. by checking whether for every pair $u, v \in V'$, edge (u,v) belongs to E.

We will reduce 3-CNF-SAT to clique problem.

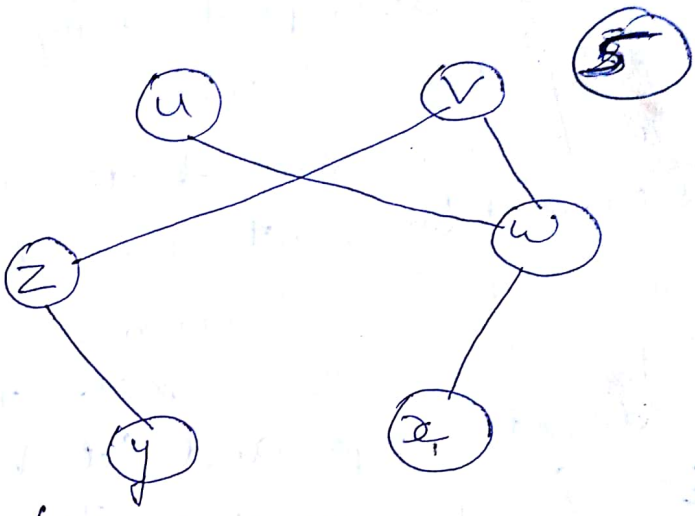
Let $\phi = C_1 \wedge C_2 \wedge \cdots \wedge C_k$ be boolean formula in 3-CNF with k clauses.

For $r=1,2,\ldots k$ for each clause C_r has exactly 3 distinct literals l_1^r, l_2^r, and l_3^r

The graph $G=(V,E)$ is constructed as follows. For each clause $C_r = (l_1^r \vee l_2^r \vee l_3^r)$ in ϕ, we place a triple of vertices v_1^r, v_2^r and v_3^r in V.



(a) An undirected graph $G=(V,E)$ with clique $V'=\{u,v,x,y\}$



(b) Graph G_1 produced by reduction algo that has vertex cover $V=V'=\{w,z\}$

Subset-sum problem

In the subset sum problem, we are given a finite set $S \subset N$ and a target $t \in N$. We ask whether there is a subset $S' \subseteq S$ whose elements sum to t.

eg. if $S=\{1,4,16,64,256,1040,1041,1093,1284,1344\}$ and $t=3754$,

then subset $S'=\{1,16,64,256,1040,1093,1284\}$ is a solution.

Subset-sum problem is NP-complete

To show that subset-sum is is NP, for an instance (S,t) of the problem, we let subset S' be the certificate checking whether $t=\sum_{s \in S'} s$

can be accomplished by a verification algorithm in polynomial time.

We can now show that vertex-cover can be reduced to subset-sum problem.

- At heart of reduction is an incidence matrix representation of G. Let $G = (V, E)$ be an undirected graph and let $V = \{v_0, v_1, \ldots v_{|V|-1}\}$ and $E = \{e_0, e_1, \ldots e_{|E|-1}\}$.

- The incidence matrix of G is $|V| \times |E|$ matrix $B = (b_{ij})$ such that

$$b_{ij} = \begin{cases} 1 & \text{if edge } e_j \text{ is incident on vertex } v_i, \\ 0 & \text{otherwise} \end{cases}$$

- The set S consists of 2 types of numbers, corresponding to vertices + edges respectively.

- For each vertex $v_i \in V$, we create a positive integer x_i whose modified base-4 representation consists of leading 1 followed by $|E|$ digits

$$x_i = 4^{|E|} + \sum_{j=0}^{|E|-1} b_{ij} 4^j$$

for $i = 0, 1, \ldots |V|-1$.

- $y_j = 4^j$

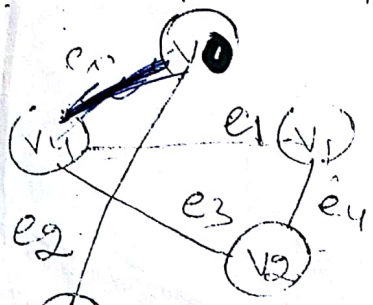
for $j = 0, 1, \ldots |E|-1$.

Suppose G_n has a vertex cover $V' \subset V$ of s.t. $V' = \{v_{i_1}, v_{i_2}, \ldots, v_{i_k}\}$

define S' by:

$S' = \{x_{i_1}, x_{i_2}, \ldots, x_{i_k}\} \cup$

$\{y_j : e_j$ is incident one precisely one vertex in $V'\}$

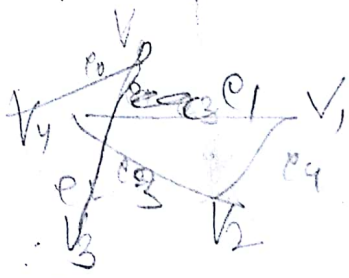


v_0
v_1
v_2
v_3

(a) Vertex cover of graph is $\{v_1, v_2, v_4\}$

(b)

	e_4	e_3	e_2	e_1	e_0
v_0	0	0	1	0	1
v_1	1	0	0	1	0
v_2	1	1	0	0	0
v_3	0	0	1	0	0
v_4	0	1	0	1	1



The corresponding incidence matrix

modified base-4

	c_5	c_4	c_3	c_2	c_1	c_0		decimal
x_0 =	1	0	0	1	0	1	=	1041
x_1 =	1	1	0	0	1	0	=	1284
x_2 =	1	1	1	0	0	0	=	1344
x_3 =	1	0	0	1	0	0	=	1040
x_4 =	1	0	1	0	1	1	=	1093
y_0 =	0	0	0	0	0	1	=	1
y_1 =	0	0	0	0	1	0	=	4
y_2 =	0	0	0	1	0	0	=	16
y_3 =	0	0	1	0	0	0	=	64
y_4 =	0	1	0	0	0	0	=	256
t =	3	2	2	2	2	2	=	3154

Subset sum instance.

In the box, is the incidence matrix.

The subset S' is marked.