NP-completeness.

We do not know too much about whether $P = NP$. Most people who have pondered the question believe that $P \neq NP$.

However, we know of a very interesting phenomenon - a host of natural problems that are complete for NP.

A decision problem X is said to be NP-Complete if

(a) $X \in NP$

(b) ~~D~~ For any $Y \in NP$,

$$Y \leq_p X.$$

If ~~only~~ (b) holds, X is said be NP-hard.

What is striking about ~~def~~ this definition is the requirement (b), which says that every problem in NP must be poly-time reducible to X. That brings up the question of whether thae are any natural problems that are NP-Complete. More on this shortly. First:

<u>Fact</u> Suppose X is an NP-complete problem. Then $X \in P$ if and only if $P = NP$.

<u>Proof</u>: Suppose $P = NP$. Then since $X \in NP$, $X \in P$.

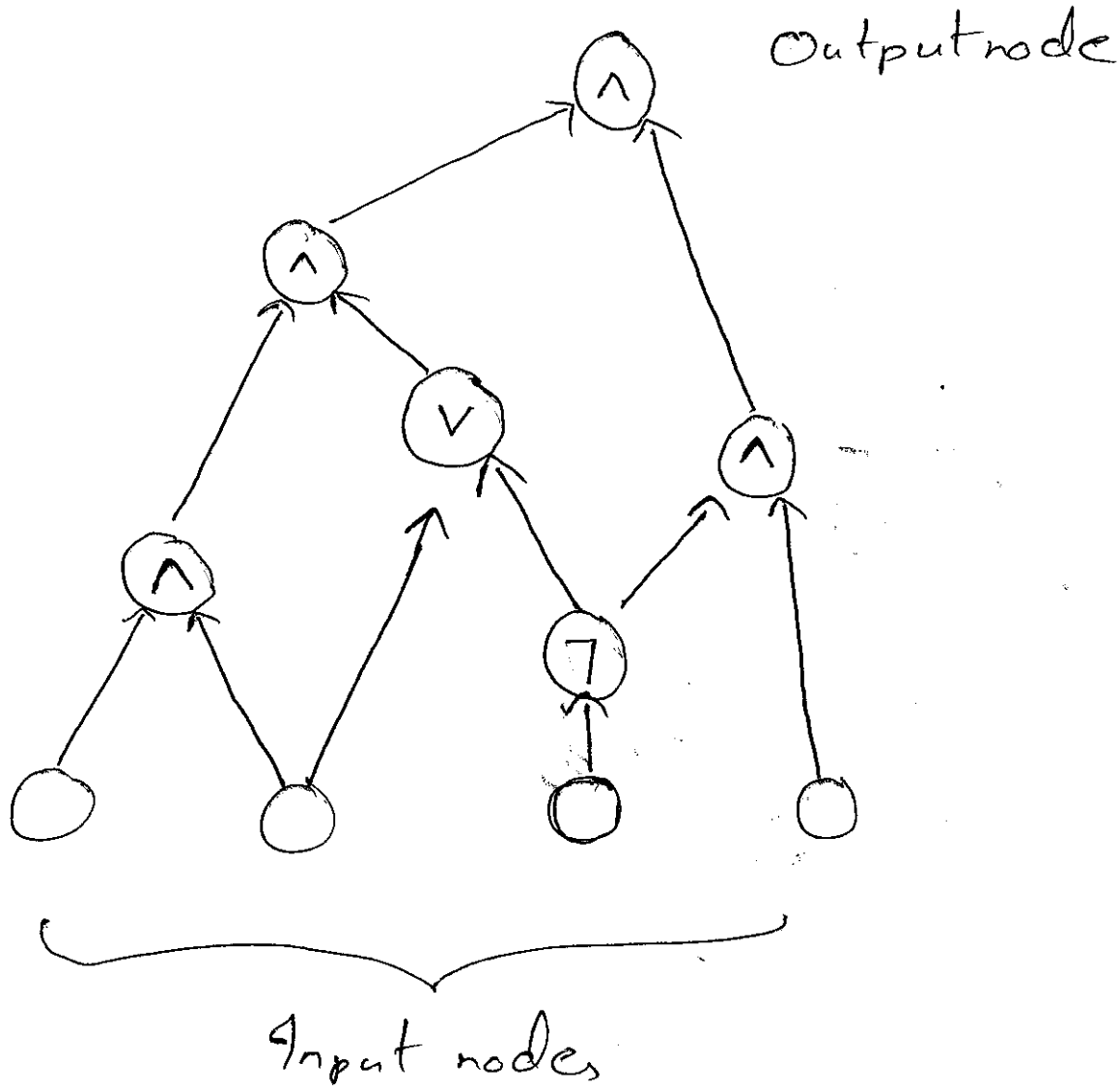
For other direction, suppose $X \in P$. Let Y be any problem in NP. Since X is NP-complete, we know $Y \leq_p X$. ~~Then~~ Since $X \in P$

and $Y \leq_p X$, $Y \in P$. We conclude that $NP \subseteq P$, and so $P = NP$.

The fact implies that if $P \neq NP$, that is, if any problem in NP can't be solved in poly-time, then no NP-Complete problem can be solved in poly-time.

We now introduce the circuit satisfiability ~~problem~~ problem, which will be our first NP-Complete problem. Here, we are given a combinatorial circuit including and/or/not gates, and we want to know if there is an assignment to input nodes that causes output to evaluate to 1.

First some examples, then the formal defns.



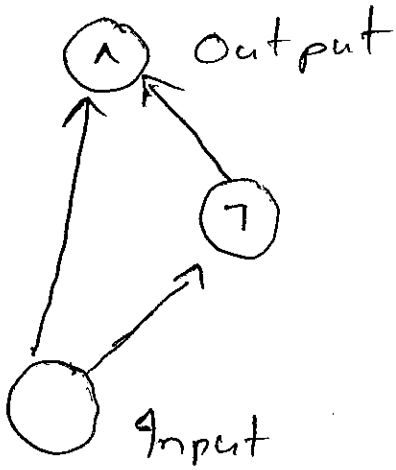
$\wedge$ - and

$\vee$ - or

$\neg$ — not

The above is a circuit. It is satisfiable — the assignment 1, 1, 0, 1 to input nodes is a satisfying assignment.

Here is a trivial circuit that is not satisfiable:



Formally, a circuit is a directed acyclic graph. The nodes with no incoming edges are ~~input~~ called input nodes. Every other node is labelled either $\wedge$, $\vee$, or $\neg$. Nodes labelled $\wedge$ or $\vee$ have two incoming edges. Nodes labelled $\neg$ have one incoming edge. There is exactly one node with no outgoing edge. This is called ~~an~~ the output node.

Since a circuit is a DAG, it can be topologically sorted. We may assume that input nodes occur first in this order. Given an assignment of T/F (or 1/0) values to input nodes, the other nodes can be evaluated as follows. We go thru the non-input nodes in order - when we arrive at node v, all nodes u such that (u, v) is an edge have been evaluated. We evaluate v as follows. Suppose v is ~~an~~ ~~and~~ labelled $\wedge$ and has incoming edges from u_1 and u_2. Then v evaluates to the "and" of whatever u_1 and u_2 evaluated to. The evaluation of v if it is labelled $\vee$ or $\neg$ is defined similarly.

Finally, the circuit evaluates to whatever the output node evaluates to.

The Circuit-satisfiability problem then is to determine, given a circuit C, whether there is an assignment to input nodes of C that causes circuit to evaluate to true.

<u>Theorem</u> Circuit-Satisfiability is NP-Complete.

<u>Proof</u> <u>Sketch</u>:

It is easy to construct an efficient verifier for circuit-satisfiability. ~~and~~ Thus the problem is in NP.

Let Y be any problem in NP.

To show $Y \leq_p$ Circuit-Satisfiability, we describe a poly-time algorithm that takes as input an instance x of Y and outputs a circuit $D(x)$ so that

x is a yes-instance of Y

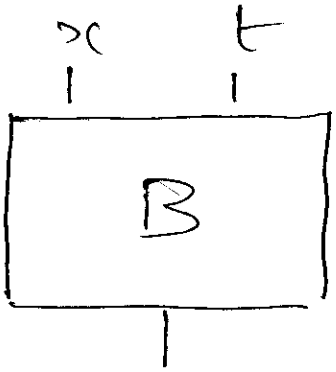
iff

$D(x)$ is satisfiable.

Our description of this algorithm will be very very sketchy.

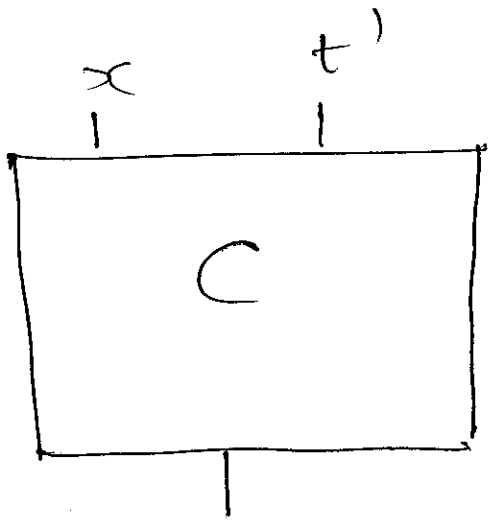
Since $Y \in NP$, it has an efficient verifier B.

We know there is a polynomial p, so that if x is a yes-instance of Y, there is a t with $|t| \le p(|x|)$ so that $B(x,t)$ outputs yes.



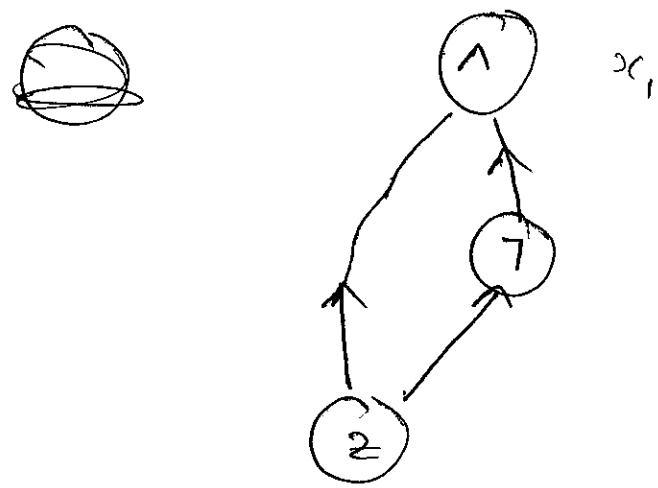
Algorithm constructs a circuit C that has "two" ~~inputs~~ sets of inputs nodes: x and t', where $|t'| = p(|x|)$. Think of t' as t followed by "end of string" pattern.

The Circuit C "simulates" B on x and t. Since B's running time is polynomial in $|x| + |t|$, and $|t| \leq p(|x|)$, B runs ~~for~~ only for a polynomial number of steps in x. So C needs to simulate B only for a polynomial number of steps. (polynomial in $|x|$). So C will have size polynomial in $|x|$.

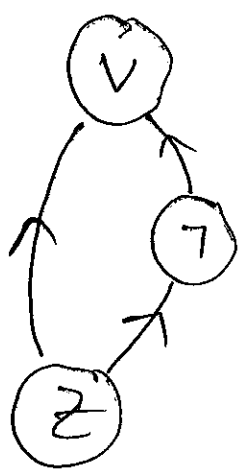


Output node of C corresponds to output of $B(x, t)$.

Finally, algo hard-codes x by adding one input node z. For example, if $x_1 = 0$, it adds

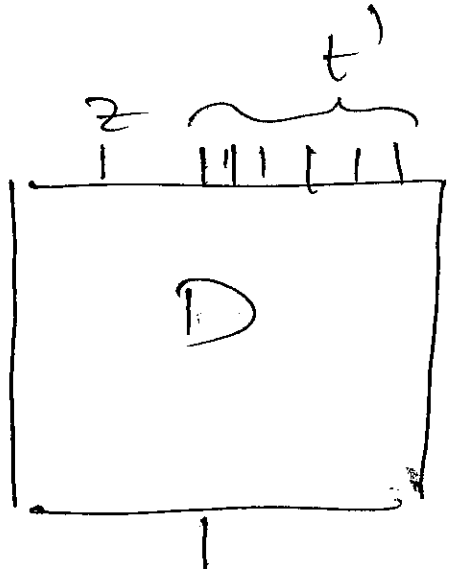


and if $x_1 = 1$, it adds



Call resulting circuit $D(x)$.

$D(x)$ has form



Since $D(x)$ simulates B on x and t,

$D(x)$ is satisfiable iff $\exists\, t$,

with $|t| \le p(|x|)$, such that

$B(x, t)$ outputs 1.

In other words,

$D(x)$ is satisfiable

iff

x is yes-instance of Y.

Transforming x to $D(x)$ is done in poly-time !!!

End of Proof Sketch.

Having one NP-complete problem makes it much easier to show other problems NP-complete. This is because of the following fact, which is easy to prove.

<u>Fact.</u> Suppose Y is NP-complete. Suppose $X \in NP$, and $Y \leq_p X$. Then X is NP-complete.

We will show Circuit-Satisfiability $\leq_p$ 3CNF-SAT. ~~Using~~ Since 3CNF-SAT $\in NP$, we conclude 3CNF-SAT is NP-Complete.

Since 3CNF-SAT $\leq_p$ IND-SET,

3CNF-SAT $\leq_p$ Vertex-Cover,

3CNF-SAT $\leq_p$ Set Cover,

and all these problems on RHS are in NP, we conclude:

IND-SET, Vertex-Cover, Set-cover are NP-complete.

To show some new problem Z to be NP-Complete, we show $Z \in NP$. This is usually easy. We pick a problem Y known to be NP-Complete. We show $Y \leq_p Z$. This can be harder, but choice of Y can greatly help us. The homework gives you some experience with this process.