

- 1a. Let k -CLIQUE be the language $\{ \langle G \rangle \mid G \text{ is a graph with a } k\text{-clique.} \}$. Show that for every fixed k , k -CLIQUE is in P .

For a graph on n nodes with fixed parameter k , there are at most $\binom{n}{k}$ possible k -CLIQUEs, which is $O(n^k)$ cliques; at most there are $\binom{n}{2}$ edges on the complete graph, which is $O(n^2)$ edges. A k -clique has $k!$ edges, which is constant for fixed k . An algorithm for k -CLIQUE is as follows:

- (a) Enumerate all possible cliques on the graph.
- (b) For each clique,
 - i. Check if all edges are present. If so, output yes and halt. Otherwise, move onto next clique.
 - ii. If no cliques found, output no.

Checking edges takes a (large) constant amount of time; the total number of times the loop is iterated in the worst case is $O(n^k)$. Hence, this k -CLIQUE $\in P$.

- 1b. Prove 7.21 by giving a poly-time reduction from CLIQUE to HALF-CLIQUE Let $\text{CLIQUE} = \{ \langle G, k \rangle \mid G \text{ is an undirected graph with a } k\text{-clique} \}$.

Let $\text{HALF-CLIQUE} = \{ \langle G \rangle \mid G \text{ is an undirected graph having a complete subgraph with at least } n/2 \text{ nodes,} \}$

We wish to give a reduction from CLIQUE to HALF-CLIQUE; that means that we give a reduction $f : x \rightarrow f(x)$ such that if $x \in \text{CLIQUE}$ then $f(x) \in \text{HALF-CLIQUE}$ and if $x \notin \text{CLIQUE}$ then $f(x) \notin \text{HALF-CLIQUE}$.

First, we note that some instances of CLIQUE map identically to HALF-CLIQUE: if we are given an instance $\langle G, k \rangle$, where $k \geq n/2$, $n = |V|$, then we simply feed that instance to HALF-CLIQUE. If that type of instance is an element of CLIQUE, then it is an element of HALF-CLIQUE.

Now consider the case where $k < n/2$. We build the following reduction: we will build a new graph $G'(V', E')$ that contains a j -sized clique, and then tune j as needed. Specifically, given an instance $\langle G, k \rangle$, we map it to $\langle G' = (V', E') \rangle$ such that G' has all of the vertices of G and j new vertices (a total of $n + j$): $V' = V \cup \{w_1, w_2, \dots, w_j\}$; further, G' has all of the edges of G and $\sum_{i=0}^{j-1} n + j - i$ new edges: $E' = E \cup \{f_1, f_2, \dots\}$ where the edges f_i connect each of the additional j vertices is connected to all of the vertices in V' .

So, G' has a j -clique by construction. To ensure that G' has a clique of size $\geq n/2$ if G has a clique of size k , we set j : since $k < n/2$, we need $k + j \geq (n + j)/2$. Therefore, $j = n - 2k$. So, in the above construction, we add $n - 2k$ nodes, and $\sum_{i=0}^{n-2k-1} 2n - 2k - i$ edges. This reduction is computable in polynomial time, since we add $O(n)$ vertices and $O(n^2)$ edges.

Now, we show that this reduction works. If G has a k -clique, then G has a $k + j$ -clique on $n + j$ nodes, which is $n - k$ -clique on $2n - 2k$ nodes, therefore G' is an instance of HALF-CLIQUE. If G' has at least an $(n + j)/2$ clique on $n + j$ nodes, then since $j = n - 2k$, G' must have had a $(2n - 2k)/2 = n - k = j + k$ clique, so G must have had a k -clique.

- 7.29 Show that if $P = NP$, a polynomial time algorithm exists that, given a Boolean formula ϕ actually produces a satisfying assignment for ϕ if it is satisfiable.

If $P = NP$, then NP complete problems could be decided in polynomial time. Hence, there would be a polynomial time decision algorithm for satisfiability of ϕ . Call this algorithm A . We can use A to build a polynomial time algorithm to compute the truth assignments for the satisfying assignment as follows:

On input of a formula ϕ , we ask A if ϕ is satisfiable. If A says no, we reject. If A says yes, then we set x_1 to TRUE inside ϕ and feed this version of ϕ to A . (Likewise, we could make a new formula ϕ' ,

where all clauses with literal x_1 are removed, and all clauses with \bar{x}_1 are modified to remove the \bar{x}_1 literal. If A says no, then we set x_1 to false, since we are confident that $x_1 = FALSE$ is part of a satisfying assignment. If A says yes, we are confident that x_1 is part of a satisfying assignment. We then continue for all variables x_i . Since there can never be more variables than clauses, at most, we may have to walk through the whole boolean formula once for each clause. Hence, in total, this takes $O(n^2)$ time, hence is a polytime algorithm. This technique is often referred to as self-reducibility.

- 7.37 A 2cnf=formula is an AND of clauses, where each clause is n OR of at most two literals. Show $2SAT \in P$.

Note that a clause of the form $(x_i \vee x_j)$ is logically equivalent to the statement $(\bar{x}_i \leftarrow x_j)$ (and of course, its contrapositive: $(\bar{x}_j \leftarrow x_i)$). Further, note that a 2SAT formula is satisfiable except when a set of clauses leads to a contradiction. For example, $(x_1 \vee x_2) \wedge (x_1 \vee \bar{x}_2) \wedge (\bar{x}_1 \vee x_2) \wedge (\bar{x}_1 \vee \bar{x}_2)$ is unsatisfiable because for every possible assignment of truth values to x_1 and x_2 , there is some clause present that contradicts another clause. In terms of implications, this means that $\bar{x}_1 \leftarrow x_2 \leftarrow x_1$ and $x_1 \leftarrow x_2 \leftarrow \bar{x}_1$. Hence, contradiction.

We will use this set of implications based on clauses to create a graph as follows: for each variable in ϕ , make 2 nodes in G , one for each literal. For every clause $(x_i \vee x_j)$, add a directed edge from \bar{x}_i to x_j and an edge from \bar{x}_j to x_i . Once the graph is built, a satisfying assignment for ϕ corresponds to a graph which contains no cycles containing both positive and negative literals, other than the trivial ones (from x_i to \bar{x}_i). Searching the graph for cycles can be done in polynomial time. In other words, you are searching for a path from some literal x_i to \bar{x}_i and from \bar{x}_i to x_i . If both chains exist, then there is a contradiction, and then there is no satisfying assignment. Else, there is a satisfying assignment. Paths can be searched for in polynomial time.