L = { anbn | n is a natural number}

Pumping Lemma: For any regular language L:

There exists a p (some number) such that for any inputs to our machine of size larger than p,

We can split the input into xyz. |xy| < p, |y| >= 1.

Consider w = apbp

X and y are all a’s.

We apply the pumping lemma:

1. For all i., xyiz is in the language.

Consider i = 0, consider i = 2. In the first case, we have less than n a’s and n b’s, in the second case we have more than n a’s and n b’s. In both cases, we do not have the string in the language and this violates the pumping lemma.

Pumping lemma for context free languages.

For any context free L:

There exists a p, such that for any inputs to our machine of size larger than p,

We can split the input into uvxyz with

|vxy| < p

|vy| > 0

All strings of the form uvixyiz are in L, for any i

Each piece u, v, x, y, z are arbitrarily long substrings

L = {anbncn | n is a natural number} w = apbpcp xvy is either all a’s, a’s followed by b’s, all b’s, b’s followed by c’s, or all c’s.

P = the class (set) of languages that can be decided by a (deterministic) Turing machine that runs in time polynomial on the size of the input.

Ex: L be a language in P, M be polytime Turing machine for L.

Input w, tM(w) = O(|w|k) for some natural number k.

TIME(f(n)) = languages decided by a single tape deterministic Turing machine in time O(f(n)).

NP = the class (set) of languages that can be decided by a non-deterministic Turing machine that runs in time polynomial in the size of the input.

Ex: L be a language in P, M be polytime nondeterministic Turing machine for L.

Input w, tM(w) = O(|w|k) for some natural number k.

A verifier is a machine that an input w, and has access to a witness or advice string c.

If a deterministic polynomial time machine can verify a language then the language is in NP.

Given a solution to a problem, can we verify the solution in polynomial time:

Our verifier is given the string w plus a “witness” (i.e. a solution) that the string w is in L. Then the verifier must check in polynomial time that w is in fact in L.

(If someone gives you a solution to a problem, can you verify the solution quickly?)

Ex: CLIQUE: Given a graph G and a number k. Is there a subset of vertices of G of size >= k such that all of those vertices are connected to each other?

L { <G,k> | there exists a subset of k vertices of G that are all connected to each other}

CLIQUE is in NP.

Suppose we are given a “solution” to the CLIQUE problem. We are given a subset of k vertices. We need to do the following in polynomial time (polynomial in |<G,k>|:

* Check that the k vertices are all in G
* Check each pair of the k vertices to see that there is an edge between them.

SUBSET SUM: We are given n integers (represented in binary) and target integer T (represented in binary). Is there a subset of the n integers that sums to exactly T?

SUBSET SUM is in NP.

A witness (a proposed solution) is a set of k numbers. We have to check that the k numbers are in the set of numbers O(kn log T)

We have to sum the numbers to see of the sum to T. Addition is linear in the number of digits of each number. O(k log T), k < n.

Polynomial time Reductions:

A <=P B if there exists a polynomial time function f: A -> B. And x is in A if and only if f(x) is in B.

F is a “preprocessor” on the input to A. We have a black box solver for B. Use the black box solver and f to solve A.

3-SAT <= CLIQUE:

3-SAT: We have n variables that can true or false. We have m “clauses” of size 3. Each clause is 3 literals. A literal is a variable x or the “not” x.

(x or (not y) or z) and ((not x) or w or (not b)) and …

(x or (not y) or z) is one clause of 3 literals. x, (not y), z

Given the clauses, is there an assignment to the variables such that every clause is true?

Turn each literal into a vertex. Connect literals in different clauses if they can be set to true at the same time.

Is this conversion (my function f) in polynomial time. Polynomial in the size of the 3-SAT problem.

I am creating 3m vertices (one for each literal, m clauses).

I am creating at most (3m)2 edges. The conversion is polynomial time.