1. Nondeterministic Algorithms
   - Verifiers

2. The Class NP

3. Nondeterministic Programming
   - The COMPOSITES Problem
   - The CLIQUE Problem
   - The VERTEX-COVER Problem
   - Other Problems

4. Optimization Problems

5. Assignment
Nondeterministic Algorithms

- When we solve a problem nondeterministically in polynomial time, we do not just magically write down the solution.
- We must be able to
  - Generate a solution in polynomial time, and
  - Verify the solution in polynomial time.
The COMPOSITES Problem

Given an integer $m$, determine whether it is composite.
The COMPOSITES Problem

- How do we generate a solution in polynomial time?
- Let $n$ be the length of the integer $m$, in bits.
- To choose a divisor $a$, choose $n$ bits to create the binary representation of $a$.
- Then do the same for $b$.
- That can be done in polynomial time (in fact, linear time).
The COMPOSITES Problem

- How do we verify that $m = ab$?
- We just multiply $a$ times $b$ and compare to $m$.
- That too can be done in polynomial time.
Outline

1. Nondeterministic Algorithms
   - Verifiers

2. The Class NP

3. Nondeterministic Programming
   - The COMPOSITES Problem
   - The CLIQUE Problem
   - The VERTEX-COVER Problem
   - Other Problems

4. Optimization Problems

5. Assignment
Verifiers

Definition (Verifier, Certificate)

A **verifier** for a language $A$ is an algorithm $V$ such that

$$A = \{ w \mid V \text{ accepts } \langle w, c \rangle \text{ for some string } c \},$$

where $c$ is a string, called a **certificate**, containing additional information needed for verification.
The language of the COMPOSITES problem is 
\{\langle m \rangle \mid m \text{ is composite}\}.

The verifier is a Turing machine that accepts \langle m, a, b \rangle whenever 
m = ab.

The certificate is \langle a, b \rangle.
Definition (The class NP)

Let $\text{NTIME}(t(n))$ be the set of all languages that are decidable nondeterministically in time $O(t(n))$. Then

$$\text{NP} = \bigcup_{k=0}^{\infty} \text{NTIME}(n^k).$$

- The class NP is the class of all languages whose members can be generated in polynomial time and that have polynomial-time verifiers.
1. Nondeterministic Algorithms
   - Verifiers

2. The Class NP

3. Nondeterministic Programming
   - The COMPOSITES Problem
   - The CLIQUE Problem
   - The VERTEX-COVER Problem
   - Other Problems

4. Optimization Problems

5. Assignment
We will assume that we have available the following three functions.

- `accept()` outputs "accept" and exits the function.
- `reject()` outputs "reject" and exits the function.
- `choose()`

The function `accept()` outputs “accept” and exits the function. The function `reject()` outputs “reject” and exits the function.
The function \texttt{choose()} takes a set $S$ as its parameter.

\texttt{choose(S)} will return a single value from the set $S$.

If there is a choice of values from $S$ that will eventually lead the algorithm to call \texttt{accept()} and a choice that leads it to call \texttt{reject()}, then \texttt{choose()} will choose at random a value that leads to \texttt{accept()}. 
The `choose()` Function

- If there is no such choice, i.e., every choice leads to `accept()` or every choice leads to `reject()`, then `choose()` will choose a random value from the set $S$.
- The execution time of `choose(S)` is $O(|S|)$.
Outline

1. Nondeterministic Algorithms
   - Verifiers

2. The Class NP

3. Nondeterministic Programming
   - The COMPOSITES Problem
   - The CLIQUE Problem
   - The VERTEX-COVER Problem
   - Other Problems

4. Optimization Problems

5. Assignment
The COMPOSITES Problem

```c
COMPOSITES(int m) {
    int a = 0;
    int b = 0;
    // Generate values for a and b
    for (i = 1; i <= n; i++) {
        a = 2*a + choose(0, 1);
        b = 2*b + choose(0, 1);
    }
    // Verify the choice
    if (m == a*b && a > 1 && b > 1)
        accept();
    else
        reject();
}
```
This is programming as it was meant to be.
Examples of NP Problems

Some NP problems
- CLIQUE
- VERTEX-COVER
- SUBSET-SUM
- TRAVELING-SALESMAN
- SAT
- 3SAT
1. Nondeterministic Algorithms
   - Verifiers

2. The Class NP

3. Nondeterministic Programming
   - The COMPOSITES Problem
   - The CLIQUE Problem
   - The VERTEX-COVER Problem
   - Other Problems

4. Optimization Problems

5. Assignment
The CLIQUE Problem

Definition (Clique)
Given a graph $G$, a clique of $G$ is a complete subgraph of $G$.

The CLIQUE Problem
Given a graph $G$ and an integer $k$, does $G$ contain a clique of size $k$?
The CLIQUE Algorithm

The CLIQUE Problem

CLIQUE(graph G, int k)
{
  // Generate a clique
  C = {};
  for (int i = 0; i < n; i++)
    if (choose(true, false))
      Add vertex i to C;
  // Verify the clique
  if (size of C != k)
    reject();
  for (int i = 0; i < k; i++)
    for (int j = 0; j < k; j++)
      if (i != j && (C[i], C[j]) ∈ G.E)
        reject();
  accept();
}
Outline

1. Nondeterministic Algorithms
   - Verifiers

2. The Class NP

3. Nondeterministic Programming
   - The COMPOSITES Problem
   - The CLIQUE Problem
   - The VERTEX-COVER Problem
   - Other Problems

4. Optimization Problems

5. Assignment
The VERTEX-COVER Problem

Definition (Vertex cover)
Given a graph $G$, a vertex cover of $G$ is a set $S$ of vertices of $G$ such that every edge of $G$ is incident to a vertex in $S$.

The VERTEX-COVER Problem
Given a graph $G$ and an integer $k$, does $G$ have a vertex cover of size $k$?
A VERTEX-COVER Algorithm

The VERTEX-COVER Problem

VERTEX_COVER(graph G, int k)
{
    // Generate a vertex cover
    C = {};
    for (int i = 0; i < n; i++)
        if (choose(true, false))
            Add vertex i to C;
    // Verify the vertex cover
    if (size of C != k)
        reject();
    for (int i = 0; i < n; i++)
        for (int j = i + 1; j < n; j++)
            if (edge(i, j) ∈ G.E)
                {bool ok = false;
                 for (int v = 0; v < k; v++)
                     if (C[v] is incident to edge(i, j))
                         ok = true;
                    if (!ok) reject();
                accept();
    }
}
Outline

1. Nondeterministic Algorithms
   - Verifiers

2. The Class NP

3. Nondeterministic Programming
   - The COMPOSITES Problem
   - The CLIQUE Problem
   - The VERTEX-COVER Problem
   - Other Problems

4. Optimization Problems

5. Assignment
The SUBSET-SUM Problem

Given a set $S = \{a_1, a_2, \ldots, a_n\}$ of positive integers and an integer $k$, does there is a subset $\{b_1, b_2, \ldots, b_m\}$ of $S$ such that

$$b_1 + b_2 + \cdots + b_m = k$$
The TRAVELING-SALESMAN Problem

Given a set of positive integers \( \{d_{i,j}\} \), \( 1 \leq i \leq n \), \( 1 \leq j \leq n \), and a positive integer \( m \), does there exist a subset of \( n \) values \( \{d_{i_1,i_2}, d_{i_2,i_3}, \ldots, d_{i_n,i_1}\} \) such that \( i_j \neq i_k \) for all \( i \neq k \) and

\[
d_{i_1,i_2} + d_{i_2,i_3} + \cdots + d_{i_n,i_1} \leq m?
\]
The SAT Problem

Given a Boolean expression \( f(x_1, x_2, \ldots, x_k) \) in \( k \) Boolean variables \( x_1, x_2, \ldots, x_n \) and consisting of \( n \) literals (an occurrence of \( x_i \) or \( \neg x_i \)), do there exist truth values for \( x_1, x_2, \ldots, x_k \) for which \( f(x_1, x_2, \ldots, x_k) \) is true? That is, is \( f \) satisfiable?
The 3SAT Problem

This is the same as SAT except that the Boolean expression is in conjunctive normal form with exactly 3 literals in each clause.
Many practical problems are optimization problems, not decision problems.

For example, classic traveling-salesman problem asks for the shortest possible route.

The clique optimization problem asks for the largest clique.
If we can solve the decision version of a problem, then we can solve the optimization version.

Consider the traveling-salesman problem.

- Initially, let $m$ be the sum of all $d_{i,j}$, for which the answer to the decision problem is trivially “yes.”
- Replace $m$ with $m/2$ and repeat.
- Continue to replace $m$ with $m/2$ until the answer is “no” and then proceed as in a binary search.
Optimization Problems

- How can we improve substantially on the choice of the initial value of $m$ and still guarantee that the answer is “yes?”
- How would we solve the optimization version of the clique problem?
Read Section 7.3, pages 241 - 248.

Exercises to be collected Fri, Dec 7, at the beginning of class. Write C programs to solve the following problems in polynomial time using the functions `choose()`, `accept()`, and `reject()`.

**Program 1:** Solve the SUBSET-SUM problem.
**Program 2:** Solve the TRAVELING-SALESMAN problem.
**Program 3:** Solve the SAT problem, given a function $f(x_1, \ldots, x_n)$.