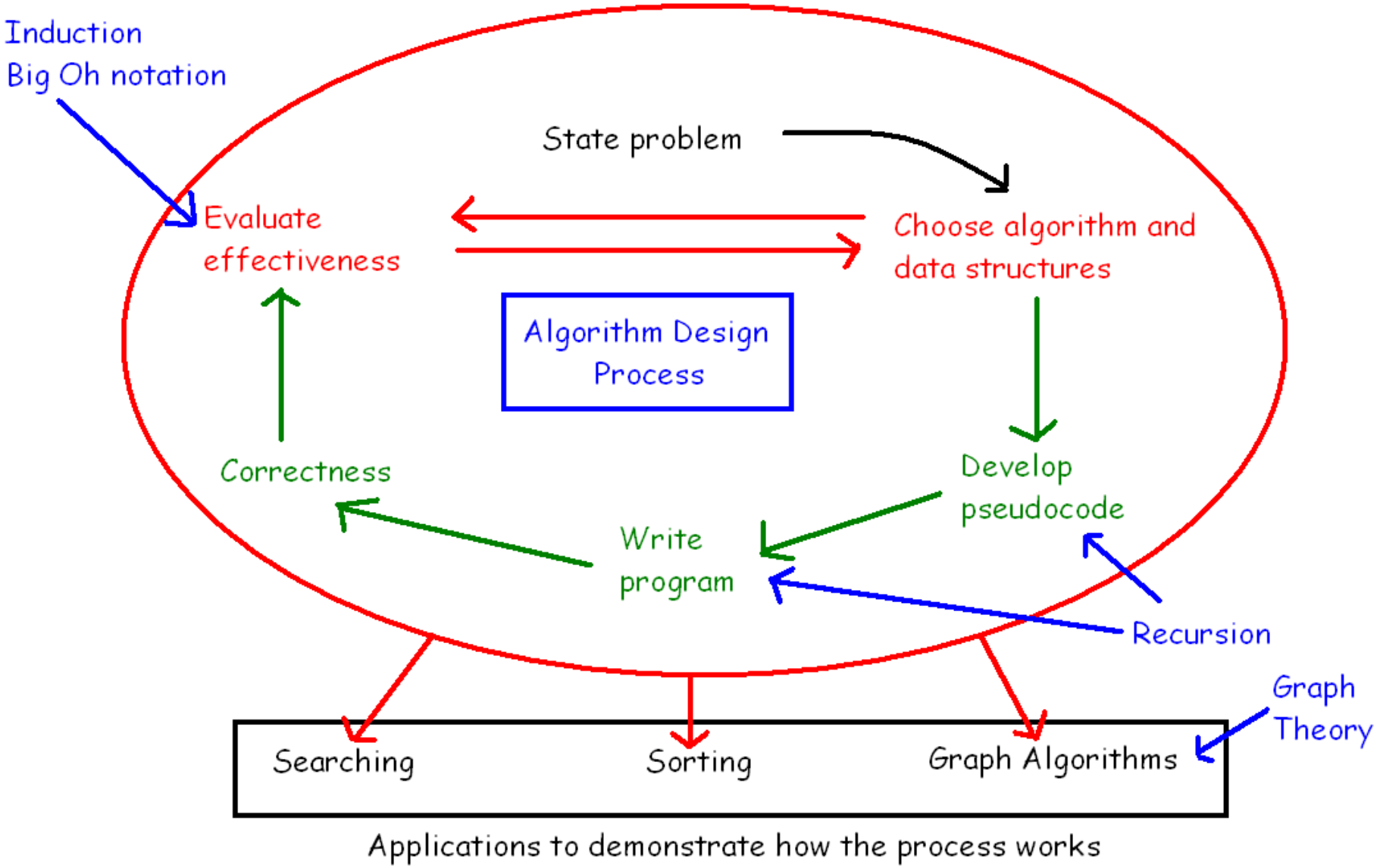

Induction
Big Oh notation
State problem
Evaluate effectiveness
Choose algorithm and data structures
Algorithm Design Process
Correctness
Develop pseudocode
Write program
Recursion
Graph Theory
Searching
Sorting
Graph Algorithms
Applications to demonstrate how the process works

Introduction to NP-completeness

All the algorithms we have studied so far run in polynomial time [$O(n^c)$ for some constant c].

There are lots of other interesting and important problems for which we do not have polynomial time solutions.

Table 1: Comparing polynomial and exponential time complexity.
Assume a problem of size one takes 0.000001 seconds (1 microsecond).

	Size *n*					
	10	**20**	**30**	**40**	**50**	**60**
n	0.00001 second	0.00002 second	0.00003 second	0.00004 second	0.00005 second	0.00006 second
n^2	0.0001 second	0.0004 second	0.0009 second	0.0016 second	0.0025 second	0.0036 second
n^3	0.001 second	0.008 second	0.027 second	0.064 second	0.125 second	0.216 second
n^5	0.1 second	3.2 second	24.3 second	1.7 minutes	5.2 minutes	13.2 minutes
2^n	0.001 second	1.0 second	17.9 minutes	12.7 days	35.7 years	366 centuries
3^n	0.059 second	58 minutes	6.5 years	3855 centuries	$2*10^8$ centuries	$1.3*10^{13}$ centuries

(from M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-completeness*, W. H. Freeman, New York, 1979.)

Table 2: Effect of improved technology on several polynomial and exponential time algorithms. The following table represents the size of the largest problem instance solvable in 1 hour.

Time Complexity function	With present computer	With computer 100 times faster	With computer 1000 times faster
n	N1	100 N1	1000 N1
n^2	N2	10 N2	31.6 N2
n^3	N3	4.46 N3	10 N3
n^5	N4	2.5 N4	3.98 N4
2^n	N5	N5+6.64	N5+9.97
3^n	N6	N6+4.19	N6+6.29

(from M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-completeness*, W. H. Freeman, New York, 1979.)

Class P

A decision problem is a yes/no question.

A decision problem is *in the class P* if there is a polynomial time algorithm for solving it.

Polynomial time: $O(n^c)$ for some constant c.

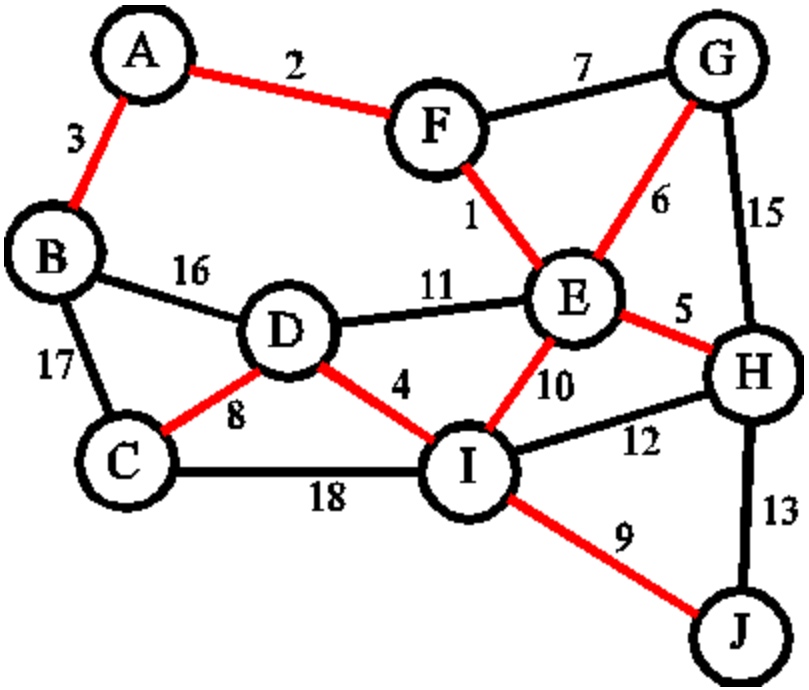
Example problem which is in P:

Minimum Weight Spanning Tree

Input: Graph G, integer k.

Question: Does G have a spanning tree of weight at most *k*?

If you are provided with a tree with weight at most k as part of the solution, the answer can be verified in $O(n^2)$ time.



Class NP

A decision problem (yes/no question) is in the *class NP* if it has a nondeterministic polynomial time algorithm. Informally, such an algorithm:

1. Guesses a solution (nondeterministically).

2. Checks deterministically in polynomial time that the answer is correct.

Or equivalently, when the answer is "yes", there is a certificate (a solution meeting the criteria) that can be verified in polynomial time (deterministically).

Hamilton Cycle is in NP: Input graph G.

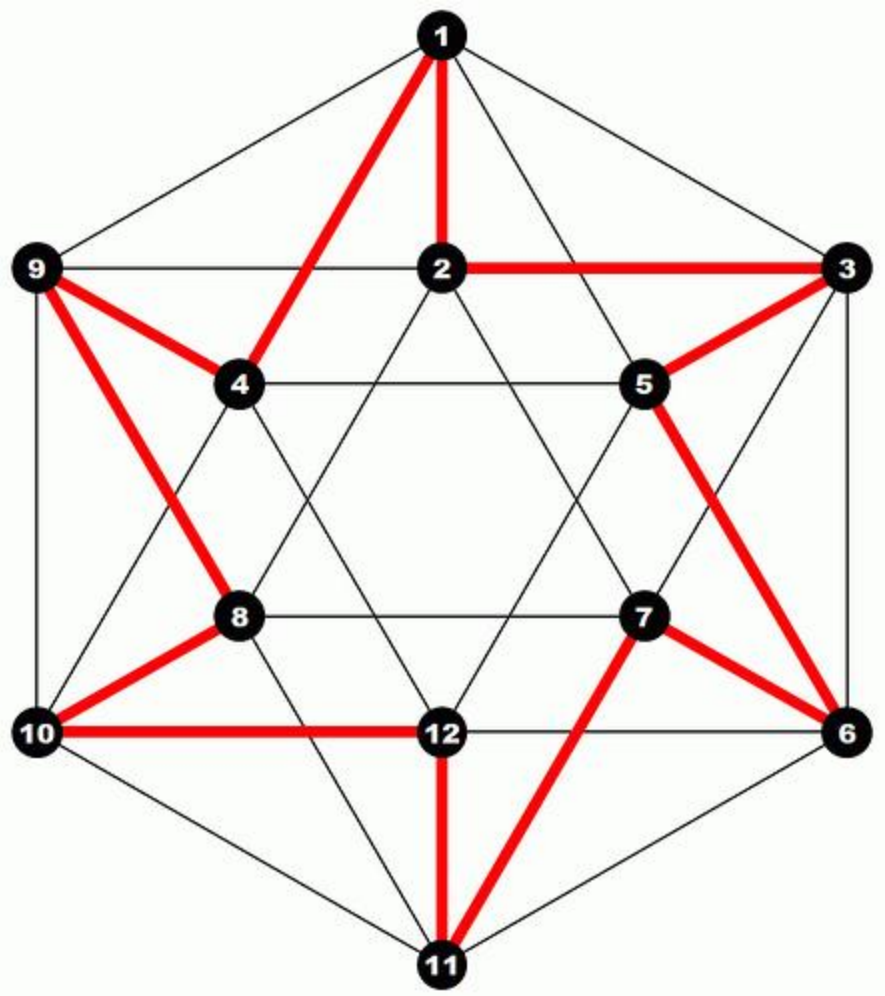
Does G have a Ham. cycle?

Certificate:

1, 2, 3, 5,

6, 7, 11, 12,

10, 8, 9, 4



Picture from: http://mathoverflow.net/faq

Hamilton Cycle is in NP: Input graph G.

Does G have a Ham. cycle?

Certificate:

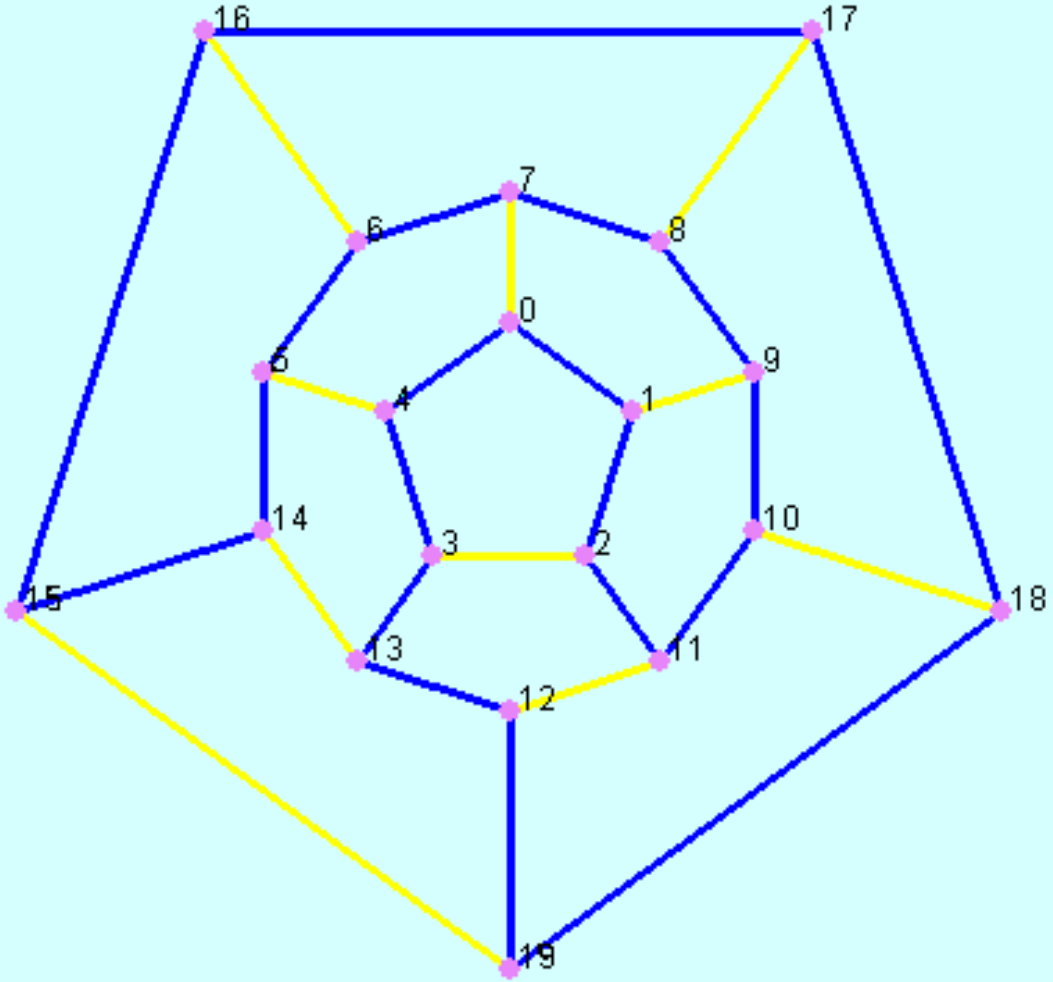
0, 1, 2, 11, 10,

9, 8, 7, 6, 5,

14, 15, 16, 17,

18, 19, 12, 13,

3, 4

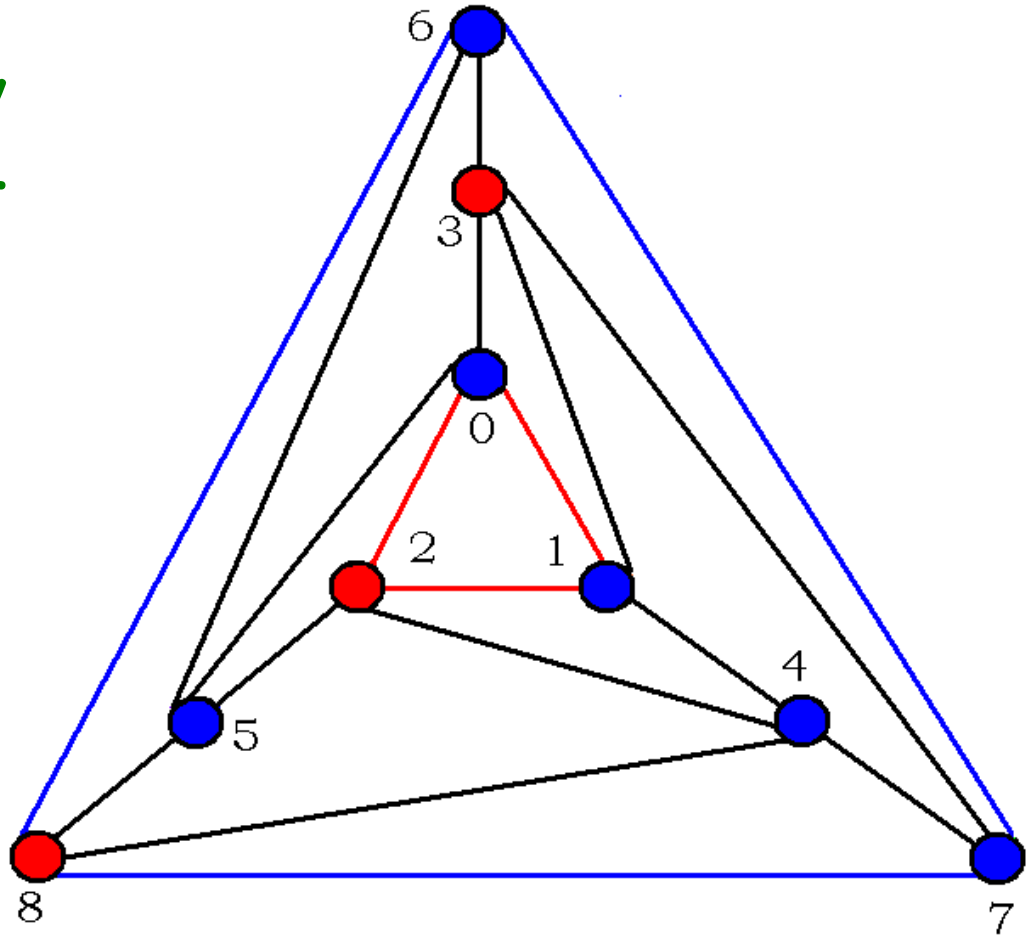


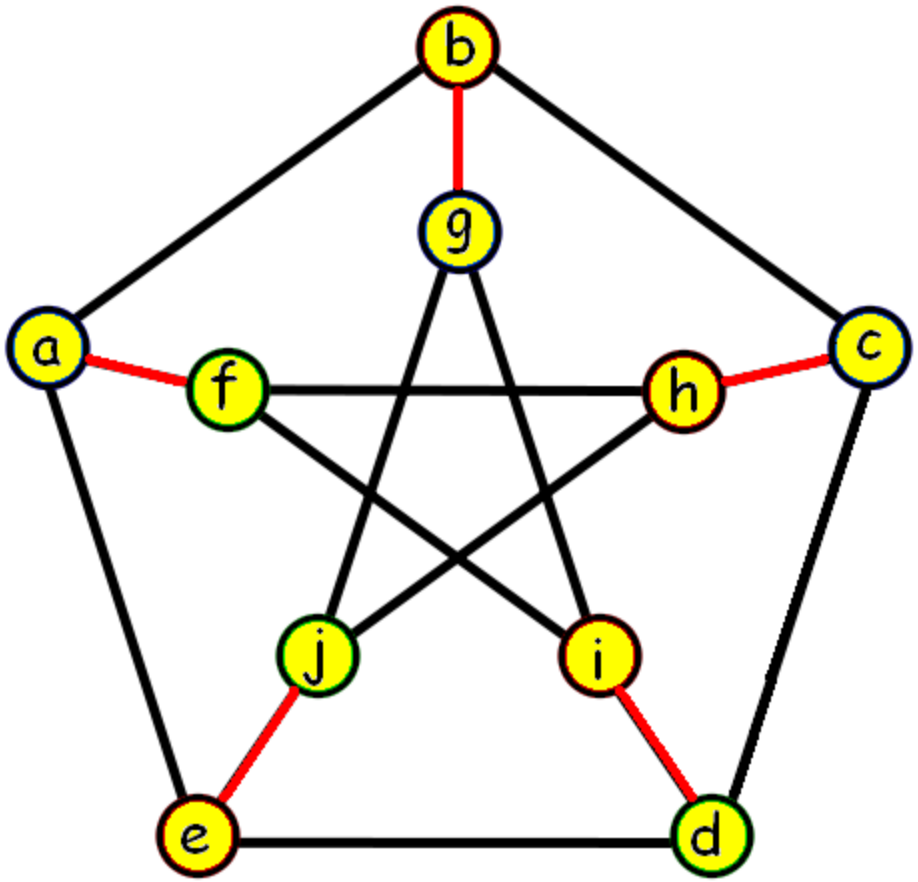
Independent Set is in NP: Given a graph G and integer k, does G have an independent set of order k?

Vertices u and v are independent if edge (u, v) is not in G.

Certificate:

2, 3, 8





Perfect matching:

(a, f)

(b, g)

(c, h)

(d, i)

(e, j)

Does P= NP? the Clay Mathematics Institute has offered a $1 million US prize for the first correct proof.

Some problems in NP not known to be in P:

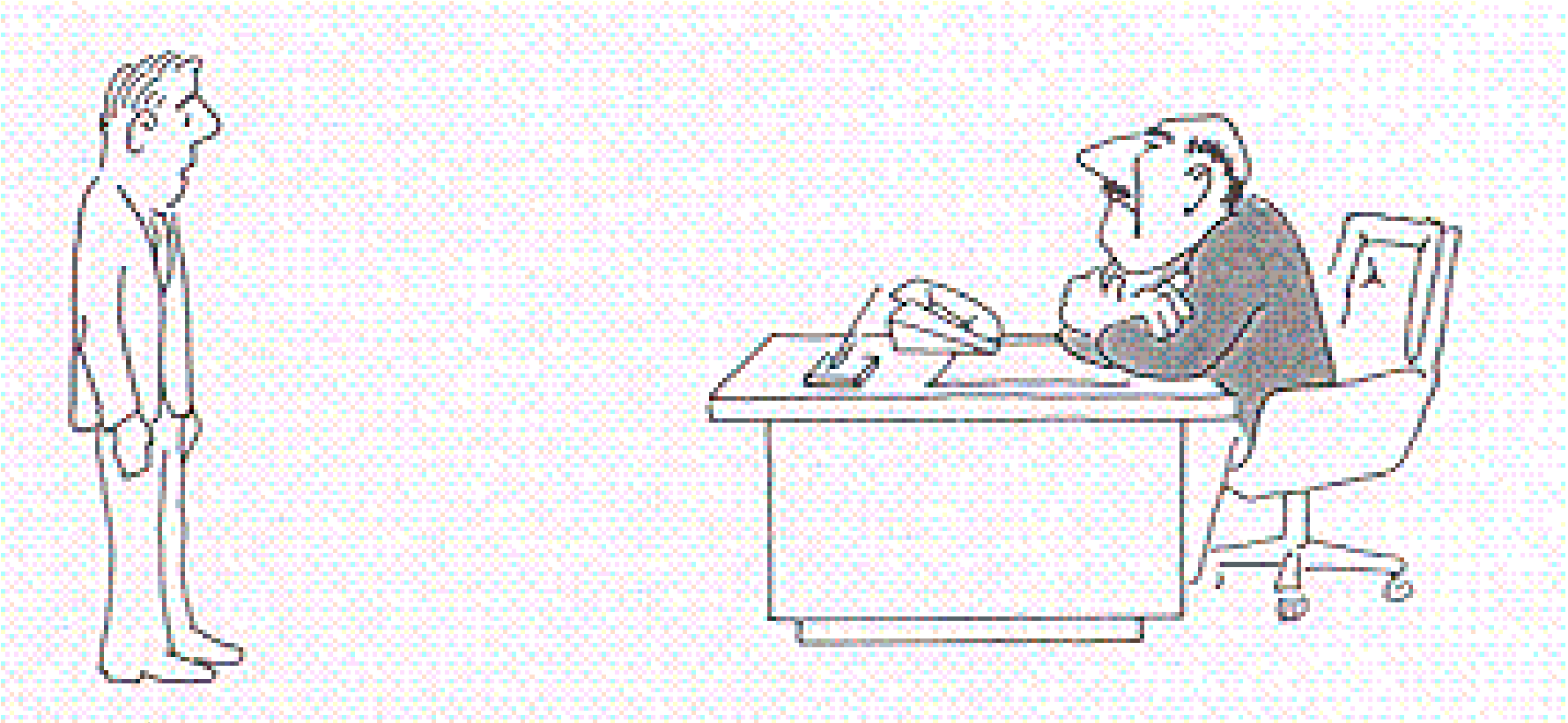
Hamilton Path/Cycle

Independent Set

Satisfiability

Note: Matching is in P. Learn more in a graph algorithms class.

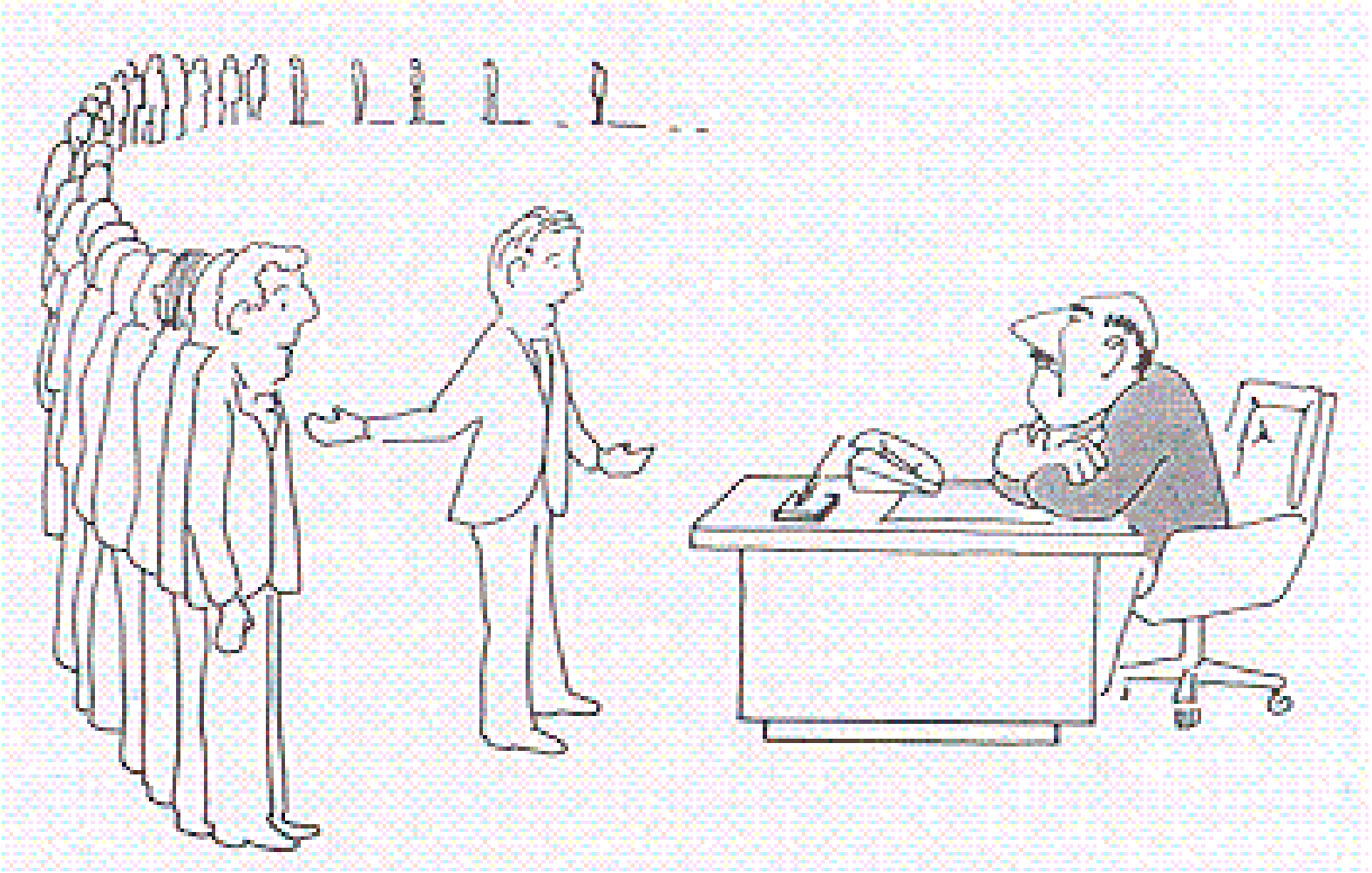
NP-completeness



I can't find an efficient algorithm,

I guess I'm just too dumb.

I can't find an efficient algorithm, because no such algorithm is possible.



I can't find an efficient algorithm, but neither can all these famous people.

NP-complete Problems

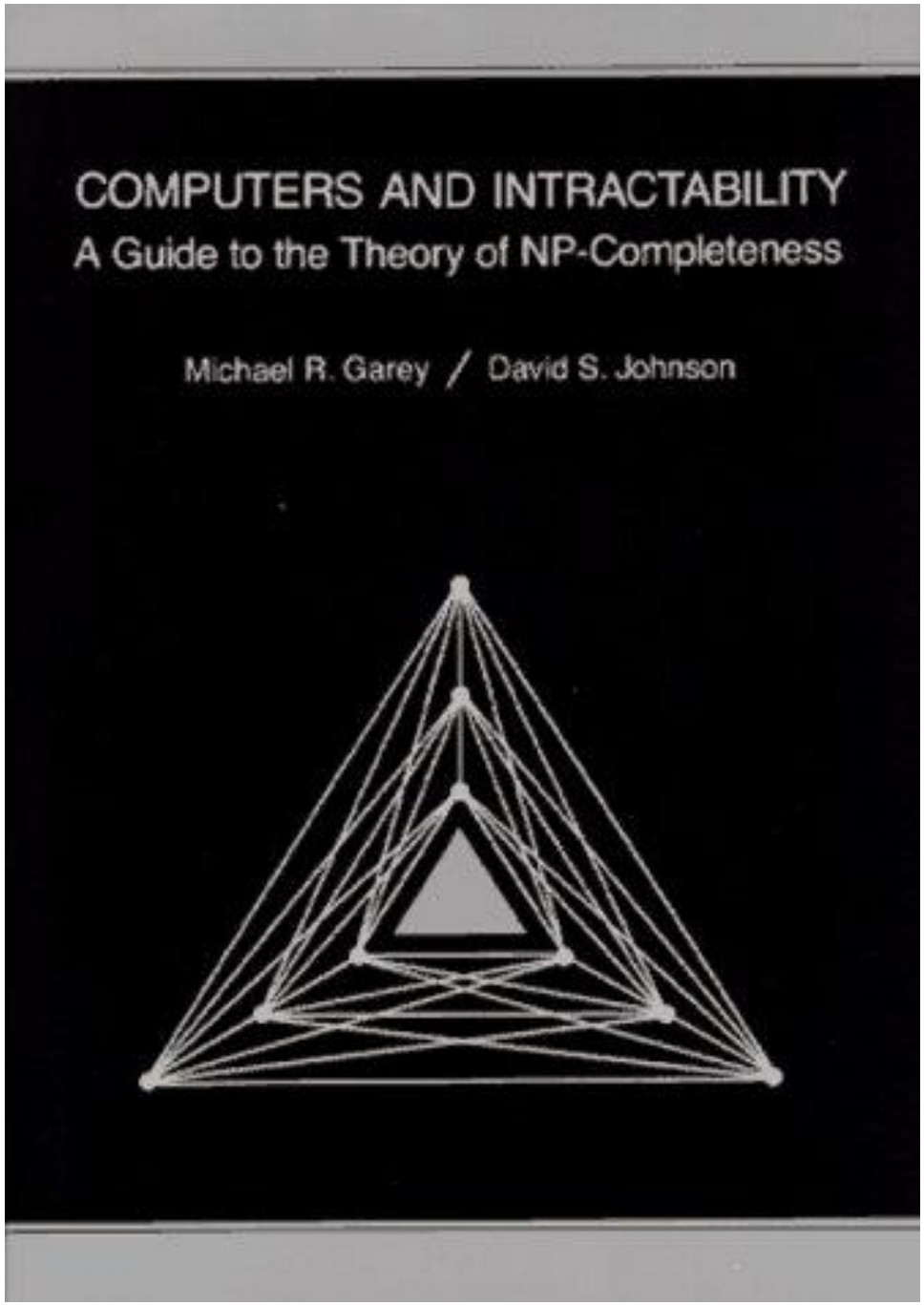
The class of problems in NP which are the "hardest" are called the *NP-complete* problems.

A problem Q in NP is NP-complete if the existence of a polynomial time algorithm for Q implies the existence of a polynomial time algorithm for all problems in NP.

Steve Cook in 1971 proved that SAT is NP-complete. Proof: you will see this if you take CSC 320 from me.

Bible for NP-completeness:

M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completness*, W. H. Freeman, 1st ed. (1979).

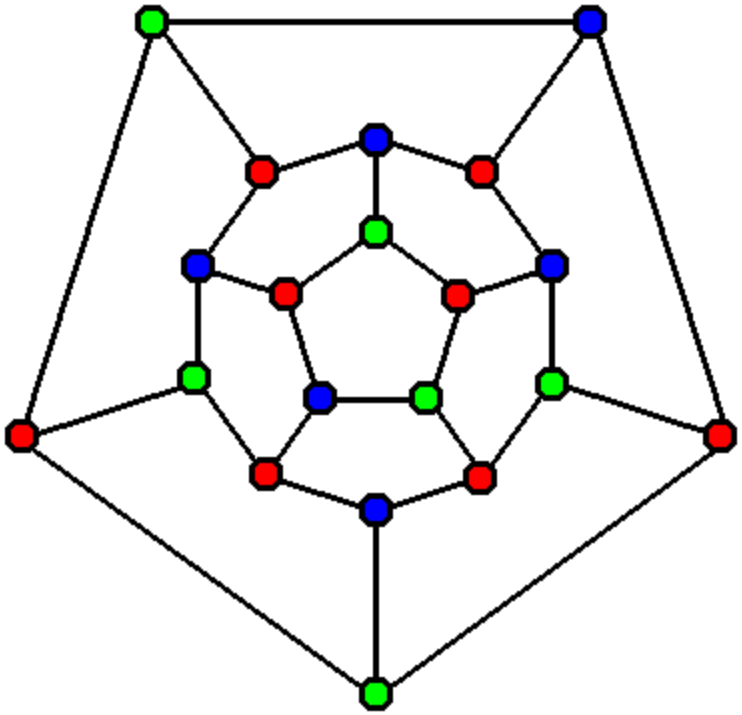


3-COLOURING: To prove this problem is in NP

Input: Graph G

Question: Does there exist a way to 3-colour the vertices of G so that adjacent vertices are different colours?

1.What could you use for a certificate for the 3-colouring problem?
2. Give the pseudo code for a polynomial time algorithm for checking your certificate.



SAT (Satisfiability)

Variables: $u_1, u_2, u_3, \ldots u_k$.

A **literal** is a variable u_i or the negation of a variable $\neg u_i$.

If u is set to *true* then ¬ u is *false* and if u is set to *false* then ¬ u is *true*.

A **clause** is a set of literals. A clause is *true* if at least one of the literals in the clause is *true*.

The input to SAT is a collection of clauses.

SAT (Satisfiability)

The output is the answer to: Is there an assignment of *true/false* to the variables so that every clause is satisfied (**satisfied** means the clause is *true*)?

If the answer is yes, such an assignment of the variables is called a **truth assignment**.

SAT is in NP: Certificate is true/false value for each variable in satisfying assignment.

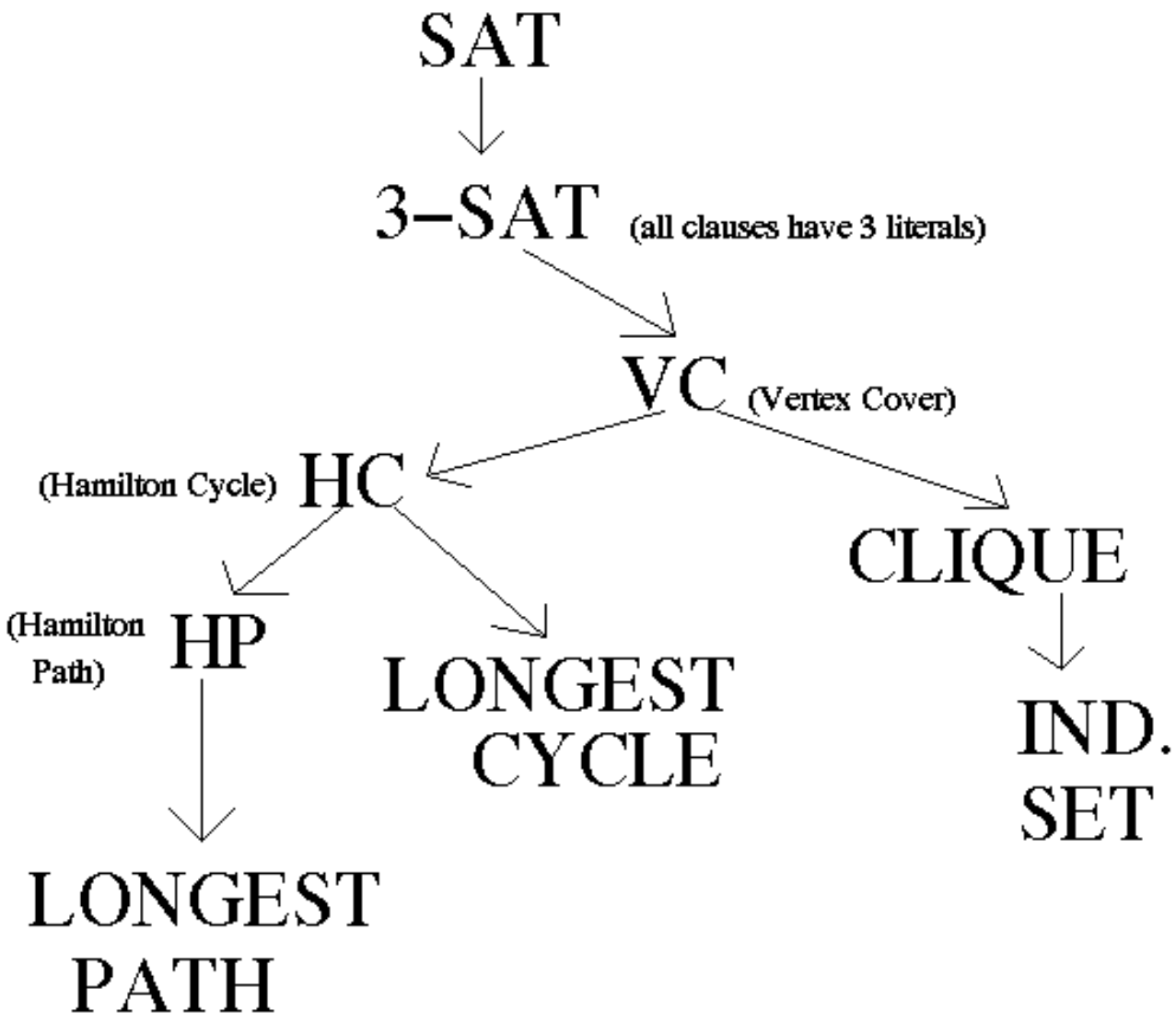
3-SAT: Every clause has exactly 3 literals.

Example of a 3-SAT Problem:

$(x \text{ OR } y \text{ OR } z) \text{ AND } (x \text{ OR } \bar{y} \text{ OR } z) \text{ AND}$

$(x \text{ OR } y \text{ OR } \bar{z}) \text{ AND } (x \text{ OR } \bar{y} \text{ OR } \bar{z}) \text{ AND}$

$(\bar{x} \text{ OR } y \text{ OR } z) \text{ AND } (\bar{x} \text{ OR } \bar{y} \text{ OR } \bar{z})$



History of NP−completeness Reductions

A set $S \subseteq V(G)$ is a vertex cover if every edge of G has at least one vertex in S.

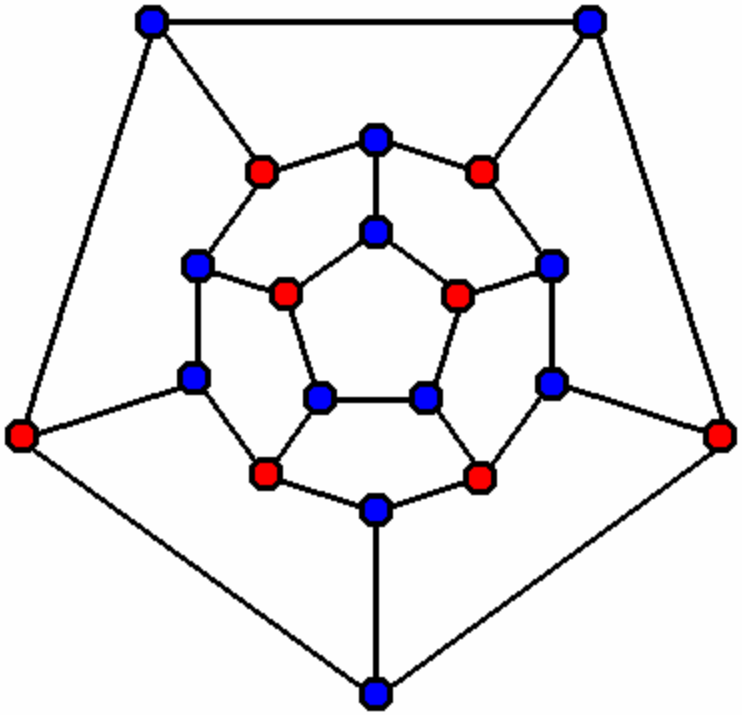
VERTEX COVER:

Given: G, k

Question: Does G have a vertex cover of order k?

Blue: vertex cover

Red: independent set



Theorem: Vertex Cover is NP-complete.

Proof: Certificate: vertex numbers of vertices in the vertex cover. To check:

```
for (i=0; i < n; i++) cover[i]= 0;
for  (i=0; i < k; i++)
{    scanf("%d", &t);
     if (t < 0 || t >= n)
          {printf("Bad cover.\n"); exit(0);}
     else  cover[t]= 1;
}
```

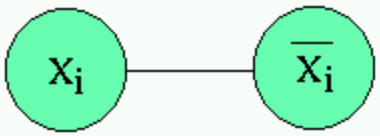
Read in certificate.

```
for (i=0; i < n; i++) {
   for (j=i+1; j< n; j++) {
       if (A[i][j]){
          if (cover[i]==0 && cover[j]==0)
          {  printf("Bad cover.\n"); exit(0); }
       }
   }
}
printf("Good cover\n");
```

Make sure each edge is covered.

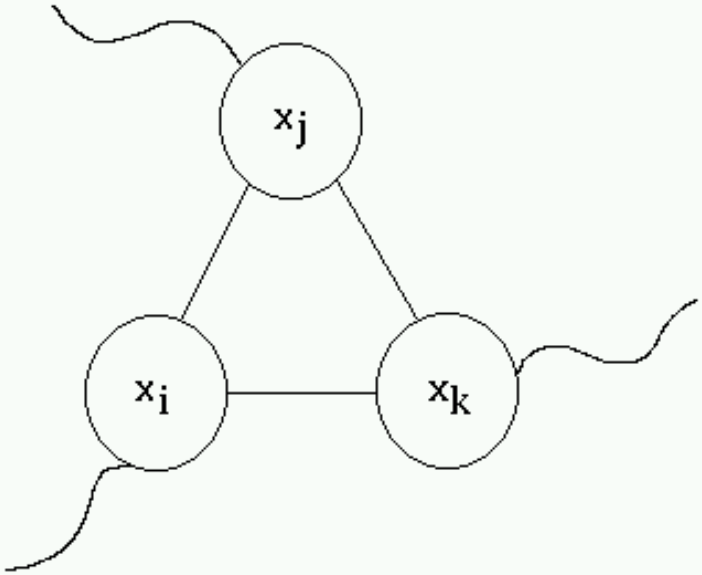
To solve 3-SAT using vertex cover:

1. For each literal x_i, include:



2. For each clause (x_i, x_j, x_k) use a gadget:

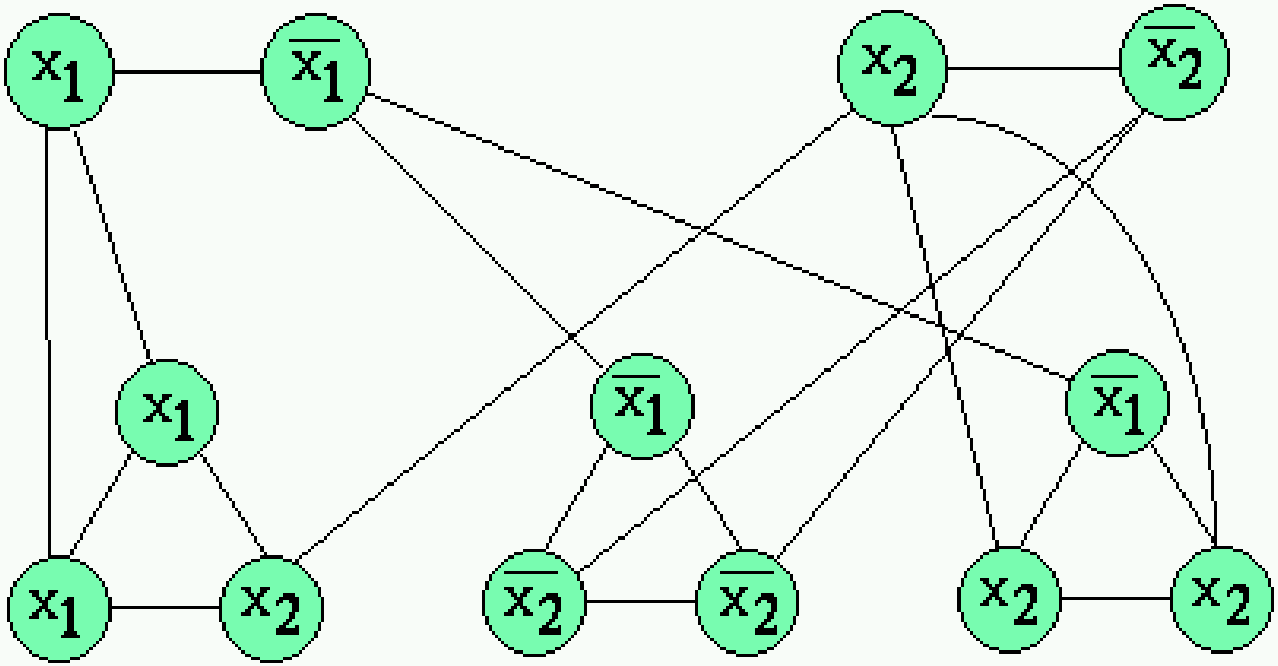
Each white vertex connects to the corresponding green one.



Pictures from:
http://cgm.cs.mcgill.ca/~athens/cs507/Projects/2001/CW/npproof.html

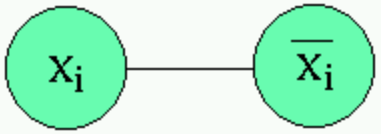
3-SAT Problem:

(x1 or x1 or x2) AND (¬x1 or ¬ x2 or ¬ x2) AND (¬x1 or x2 or x2)

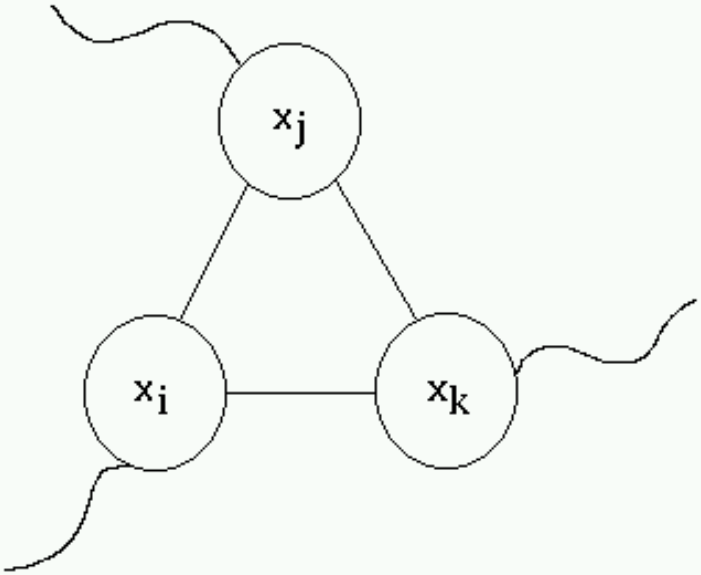


At least one vertex from

is in the vertex cover.



For each gadget, at least 2 vertices are in the vertex cover:

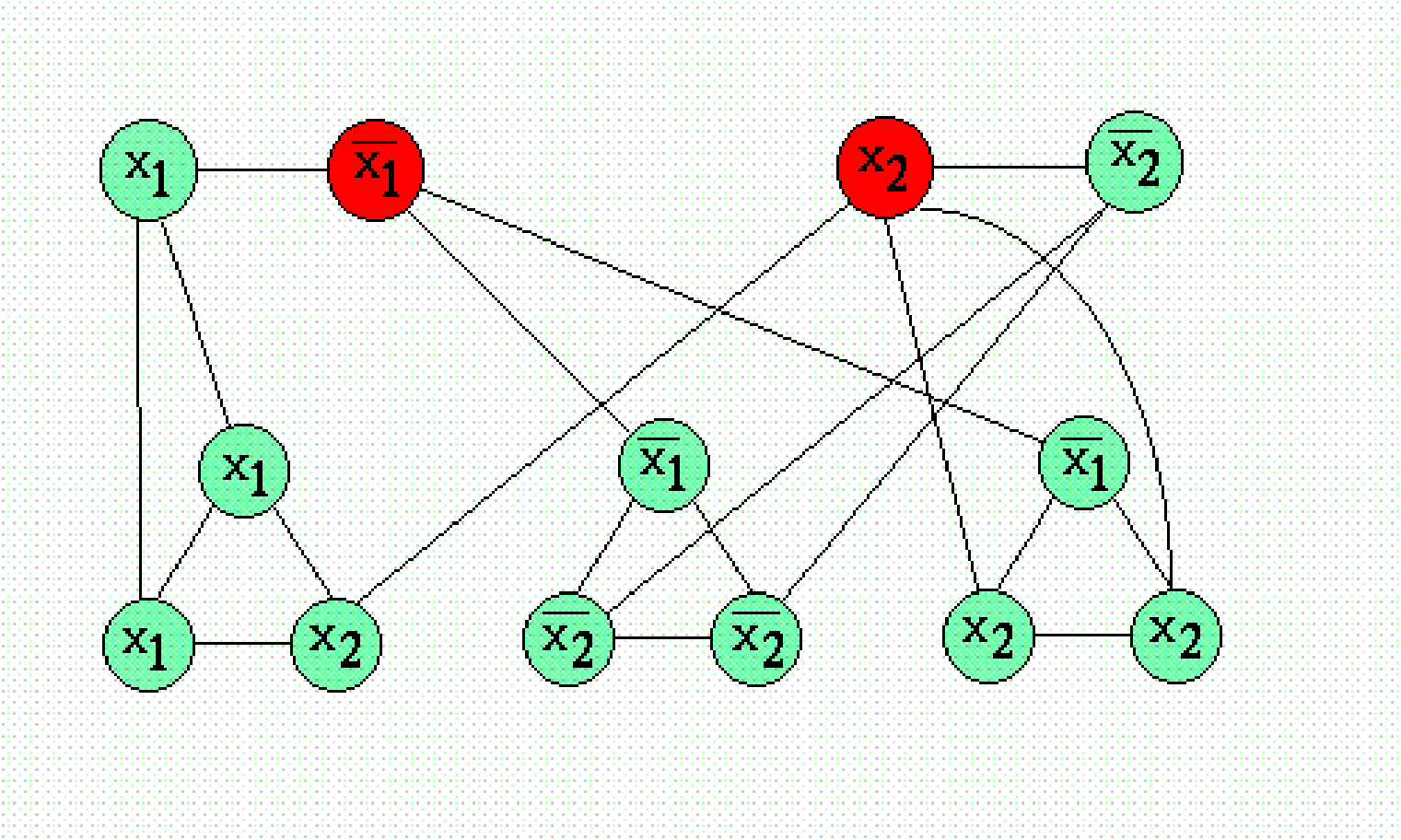


Number of variables: n

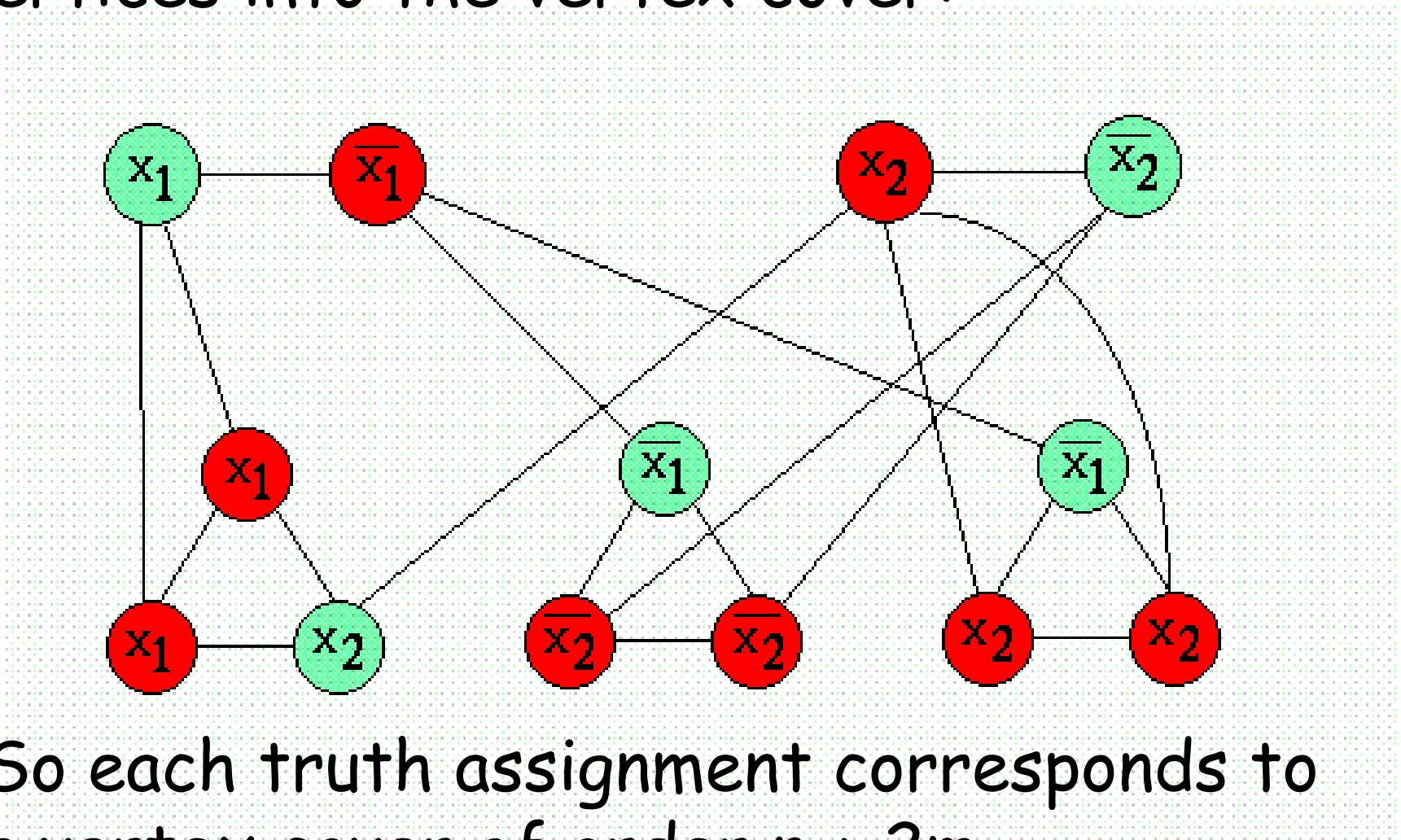
Number of clauses: m

When is there a vertex cover of order n + 2m?

Put vertices corresponding to true variables in the vertex cover.



Satisfying assignment: Each clause has at least one true variable. Put two other vertices into the vertex cover:



So each truth assignment corresponds to a vertex cover of order n + 2m.

Any vertex cover of order n + 2m corresponds to a satisfying assignment because we can only select at most one of x and ¬x (these are the true variables). The true variables must satisfy each clause since at most 2 vertices can be selected from each clause gadget.