Exercise 15.4-2

- *7:29 pm*

```
PRINT_LCS(c, X, Y, i, j)
    if c[i][j] == 0
        return
    if X[i] == Y[j]
        PRINT_LCS(c, X, Y, i-1, j-1)
        print X[i]
    elseif c[i-1][j] > c[i][j-1]
        PRINT_LCS(c, X, Y, i-1, j)
    else
        PRINT_LCS(c, X, Y, i, j-1)
```

- *7:57pm*

```
PRINT-LCS(c, X, Y, i, j)
  if c[i, j] == 0
    return
  if X[i] == Y[j]
    PRINT-LCS(c, X, Y, i - 1, j - 1)
    print X[i]
  else if c[i - 1, j] > c[i, j - 1]
    PRINT-LCS(c, X, Y, i - 1, j)
  else
    PRINT-LCS(c, X, Y, i, j - 1)
```

Exercise 15.4-2

- *9:13 pm*

```
LCS(X, C, r, j)
   if r==0 or j==0
         return // stop
   if C[r,j]==C[r-1,j-1]
         LCS(X, C, r-1, j-1)
         print(X[r]) // row number
         else if C[r-1,j]>=C[r,j-1]
               LCS(X, C, r-1, j) // go up a row
         else
               LCS(X, C, r, j-1) // go left a column
```

- *9:25pm*

```
PRINT-LCS (c, X, Y, i, j)
    if c[i, j] == 0
        return
    if X[i] == Y[j]
        PRINT-LCS (c, X, Y, i-1, j-1)
        print X[i]
    else if c[i-1, j] > c[i, j-1]
        PRINT-LCS (c, X, Y, i-1, j)
    else
        PRINT-LCS (c, X, Y, i, j-1)
```

Exercise 15.4-2

- *9:38pm*

```
PRINT-LCS(c, A, i, j)
  if i == 0 or j == 0
    return
  if c[i,j] == c[i-1, j-1] + 1     \\ equivalent to b[i,j] == 'D'
    PRINT-LCS(c, A, i-1, j-1)
    print xi
  elseif c[i,j] = c[i-1, j]         \\ equivalent to b[i,j] == 'V'
    PRINT-LCS(c, A, i-1, j)
  else                              \\ equivalent t0 b[i,j] == 'H'
    PRINT-LCS(c, A, i, j-1)
```

- *Solution*

RECONSTRUCT-LCS(c, X, Y, i, j)
if $i == 0$ or $j == 0$
    return
if $x_i == y_j$
    RECONSTRUCT-LCS$(c, X, Y, i-1, j-1)$
    print x_i
elseif $c[i, j] == c[i-1, j]$
    RECONSTRUCT-LCS$(c, X, Y, i-1, j)$
else RECONSTRUCT-LCS$(c, X, Y, i, j-1)$

NP-Completeness (Nondeterministic Polynomial Time Completeness)

Sushanth Sivaram Vallath
&
Z. Joseph

Overview

- Algorithms seen so far are $O(n^k)$
- Are all problems polynomial time?
- There are problems that cannot be solved by any computer no matter how long it takes
- There are problems that can be solved but not in $O(n^k)$
- Problems that can be solved in polynomial time are termed as tractable
- Problems that require super-polynomial time are intractable, or *hard*

Overview

- NP-complete problems are those that have no known polynomial solution. Further, no one has ever been able to prove that *no* polynomial time algorithm exists for them
- Some NP-complete problems
  - Longest path problem
  - Hamiltonian cycle problem
  - 3-CNFsatisfiability

CNF-conjunctive normal form

Polynomial (P) Problems

- Are solvable in polynomial time
- Are solvable in $O(n^k)$, where k is some constant and n is the size of the problem
- Almost all the algorithms we have covered so far are P problems

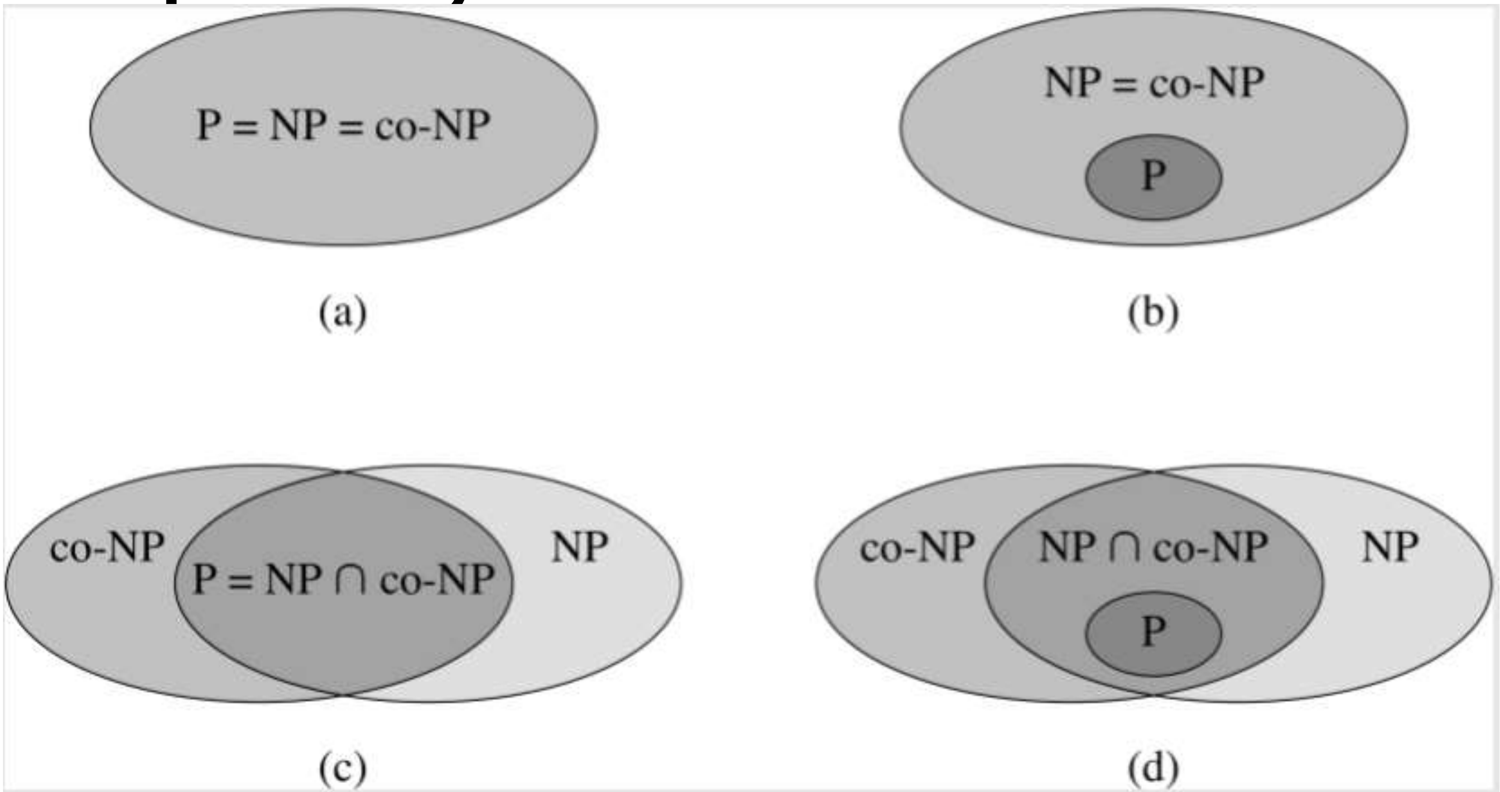
NP Problems

- This class of problems has solutions that are verifiable in polynomial time
- What is meant by “verifiable?”
  - Say a solution to a hamiltonian cycle is provided: $(v_1, v_2, v_3, \ldots, v_k)$
  - If we can easily verify in polynomial time that all (v_i, v_{i+1}) are edges of the graph and form a simple cycle then it is a NP problem
- Any problem P is also NP
- Or, P $\subset$ NP

co-NP Problems

- Say there are a set of problems L,such that L's complement $\overline{L} \in$ NP
- Then L are termed as co-NP class of problems

Complexity Classes



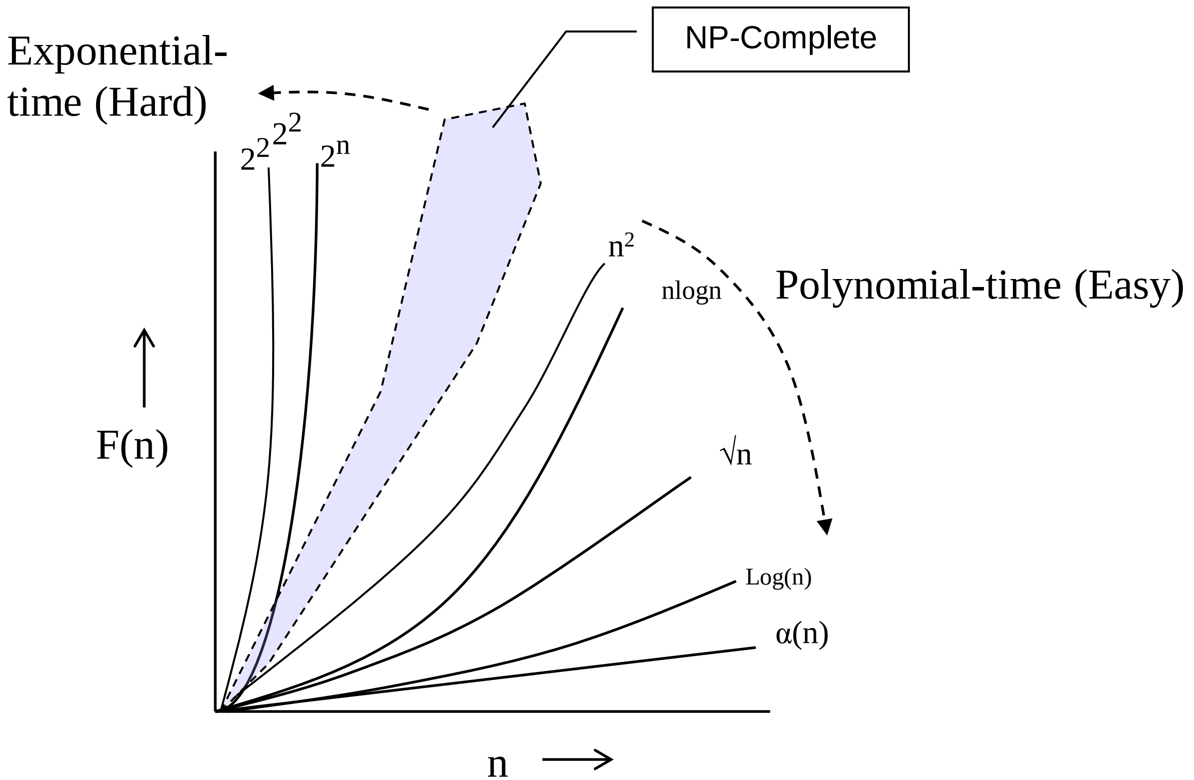
Four possible relationships for complexity classes

a) P = NP = co-NP
b) If NP is closed under complement then P=co-NP
c) P = NP∩co-NP
d) P ≠ NP∩co-NP; most researchers regard this is most likely

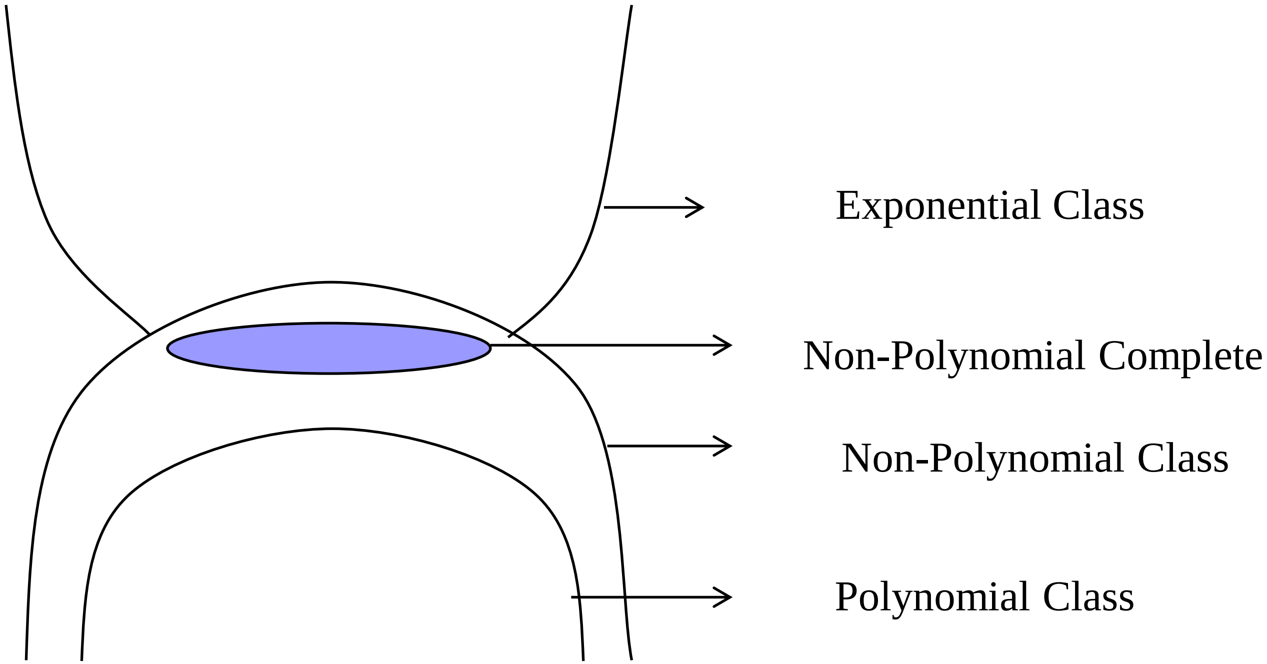
NP-Complete Problems

- Are NP problems whose polynomial-time algorithm has never been discovered
- As difficult, or hard, as any NP problem
- If any NPC problem can be solved in polynomial time then all NP problems have polynomial time algorithms
- If we can establish a problem is NPC, we provide the evidence for its intractability
- In this case we try to find an approximation algorithm, rather than searching for a fast algorithm that provides exact solution

Exponential Time Algorithms



Where does NP Complete lie?



NPC Examples

- Longest path problem: (similar to Shortest path problem, which requires polynomial time) suspected to require exponential time, since there is no known polynomial algorithm.
- Hamiltonian Cycle problem: Traverses all vertices exactly once and form a cycle.

Examples

- 3-CNF Satisfiability[1]
  - A Boolean formula is in k-conjunctive normal form, or k-CNF, if it is the AND of clauses of OR of exactly k variables
  - For example, a 2-CNF is

$$(x_1 \vee \overline{x_2}) \wedge (\overline{x_1} \vee x_3) \wedge (x_1 \vee \overline{x_2})$$

  - This function is satisfiable with $x_1 = 1$, $x_2 = 0$, and $x_3 = 1$

3-CNF sasifiability is NP-complete

[1]A bolean formula is satisfiable if there is some assignment of its variables that results in a logical 1

How to Show NPC

- To demonstrate that a problem is NPC, we are stating how hard is the problem
- Instead of finding an algorithm, we try to prove that no efficient algorithm is likely to exist
- We rely on three concepts for this demonstration:
  - Decision problems vs. optimization problems
  - Reductions
  - A *first* NPC problem

Decision vs. Optimization

Optimization problems: A solution has an associated value, we wish to find a feasible solution with best value

Ex: Several paths from u to v in a graph. SHORTEST-PATH (a feasible solution) finds a path that uses fewest edges (best value)

Decision problems: in which the answer is simply "yes" or "no"

Ex: Does a path exist from u to v consisting of at the most k edges?

NP-completeness applies to *decision problems*, and not to *optimization problems*

Converting an *optimization problem* into its related *decision problem* helps to show that the problem is "hard"

Reduction

Say

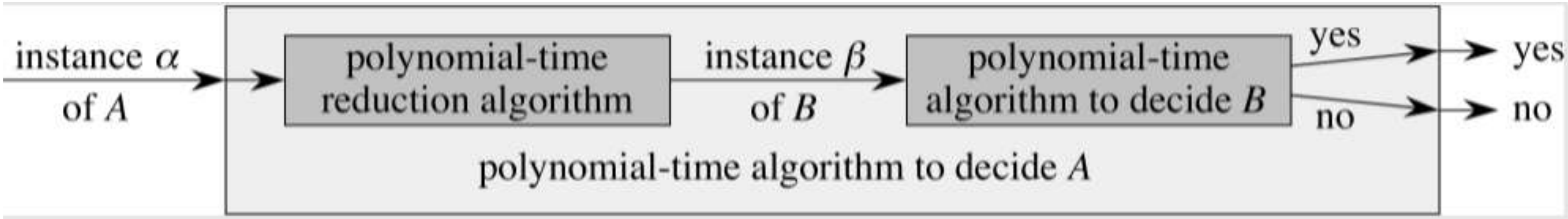
- B is a problem (easy/hard?)
- A is known to be difficult

If we can solve A using B as a subroutine then A is solvable

A reduction is an algorithm for transforming one problem (A) into another problem (B)

A sufficiently efficient reduction from one problem to another may be used to show that the first problem is at least as difficult as the second one

Reduction



- Given a polynomial-time reduction algorithm
- Given a polynomial-time decision algorithm for problem B

Solution of the instance β of B will be the solution of instance α of A

Conversely, if A is known to have no polynomial-time algorithm then no polynomial-time algorithm can exist for B

That is, if B is a subroutine of A, and A is hard to solve then B is hard to solve too

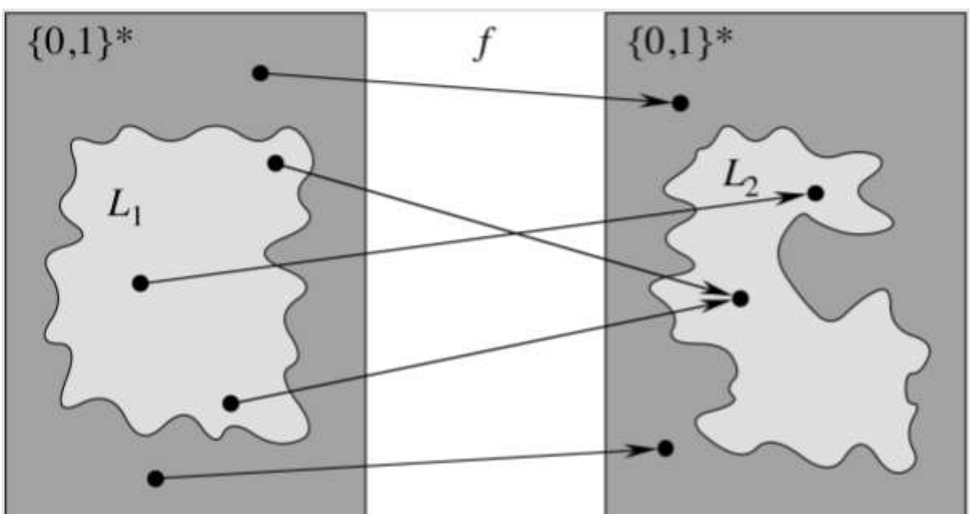
Reduction---notations

- Intuitively, a problem L_1 can be reduced to another problem L_2 if an instance of L_1 can be "easily rephrased" as an instance of L_2

- Say there is a polynomial-time function f such that for any $x \in L_1$, $f(x) \in L_2$ then f is a reduction function
- If f is a polynomial time computable function then we can write:

$$L_1 \leq_p L_2$$

which means that L_1 is polynomial-time reducible to L_2



A *first* NPC Problem

- Reduction technique relies on having a problem known to be NPC
- Circuit (or Bolean) satisfiability is used as the “first” problem
- We begin by proving that this first is NPC

A Formal-language Framework

Thinking of all problems as decision (1 or 0) problems, we can utilize formal-language theory, which is reviewed as

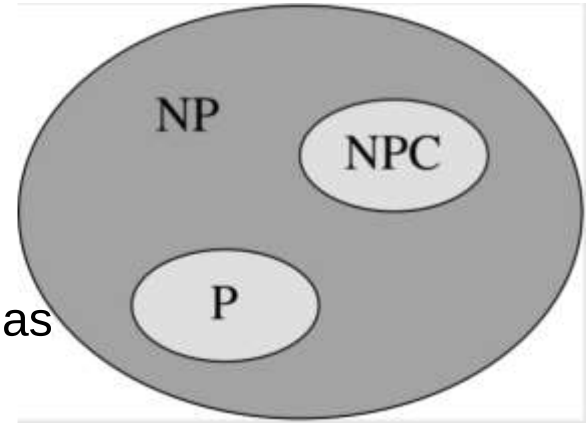
- An ***alphabet*** Σ is a finite set of symbols
- A ***language*** L over Σ is any set of strings made up of symbols from Σ
  - For example, if $\Sigma = \{0,1\}$, the set $L = \{10, 11, 101, 111, 1011, 10001, \ldots\}$ is the language of binary numbers
- An ***empty string*** is represented by ε and an empty language by $\emptyset$
- Σ^* represents the language of all strings of Σ
  - For example, $\Sigma^* = \{\varepsilon, 0, 1, 00, 10, 01, 11, 000, 001, \ldots\}$ is the set of all binary strings
  - Every language L of Σ is a subset of Σ^*

NP-complete Formal Definition

- A language $L \subseteq \{0,1\}^*$ is NP-complete if
  1. $L \in$ NP
  2. $L' \leq_{\mathrm{p}} L$ for every $L' \in$ NP
- If a language L satisfies property 2, but not necessarily property 1, we say L is NP-hard

- Theorem
  If an NP-complete problem is solvable in polynomial time then P=NP
  Equivalently, if any problem in NP is not polynomial-time solvable, then no NP-complete is polynomial-time solvable (counter-positive of the first statement)

(Proof in the book)



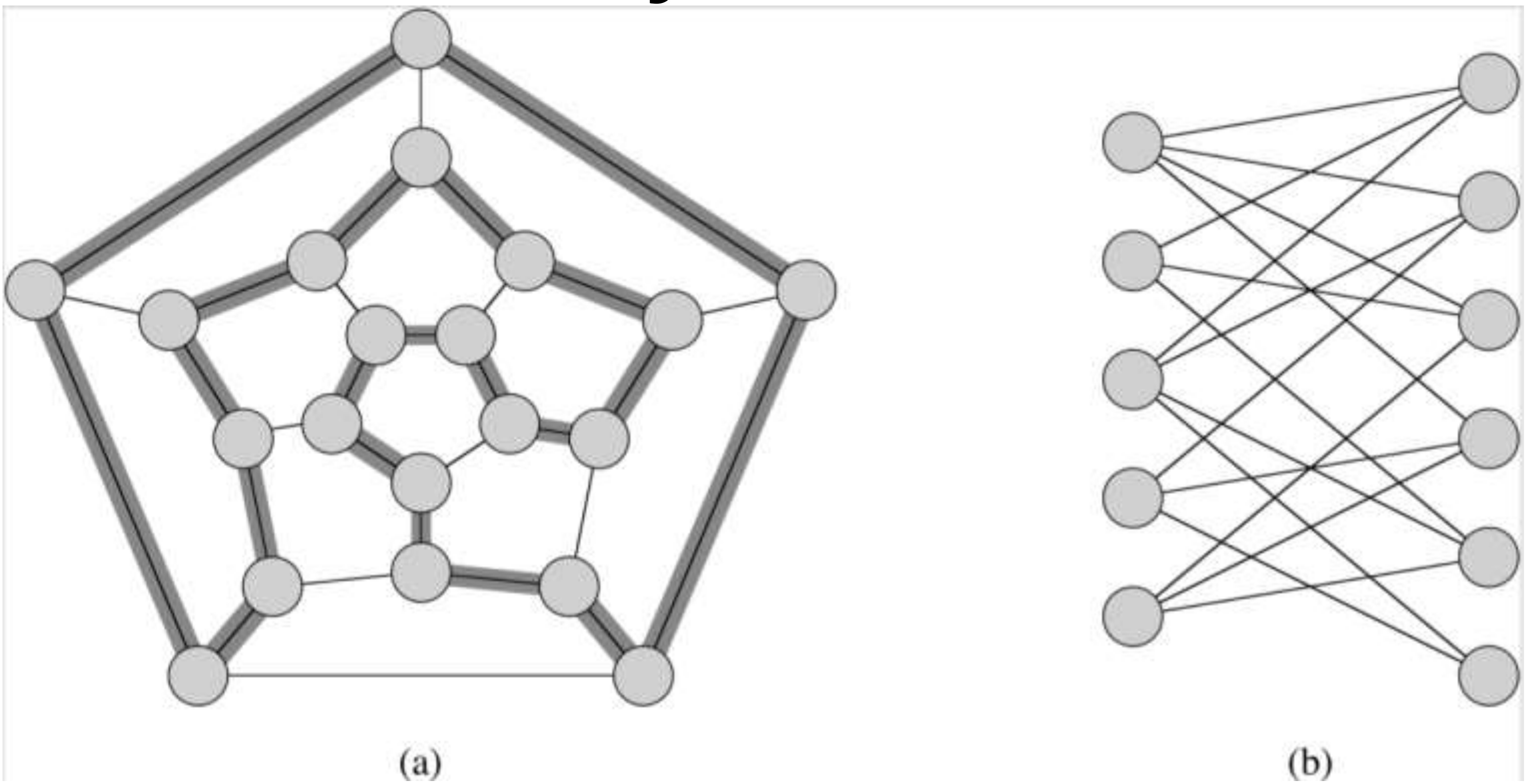
Note: Most computer scientists view the relationships as
P⊂NP, P⊂NP, and P∩NPC= ∅

Hamiltonian Cycles

- A simple cycle in an undirected graph that contains each vertex of the graph
- The name honors W. R. Hamilton, who described the game shown on next slide
- One player sticks five pins in five consecutive vertices
- Other player must complete the path to form a cycle containing all the vertices

4 August 1805 – 2 September 1865) was an Irish mathematician. While still an undergraduate he was appointed Andrews professor of Astronomy and Royal Astronomer of Ireland

Hamiltonian Cycles



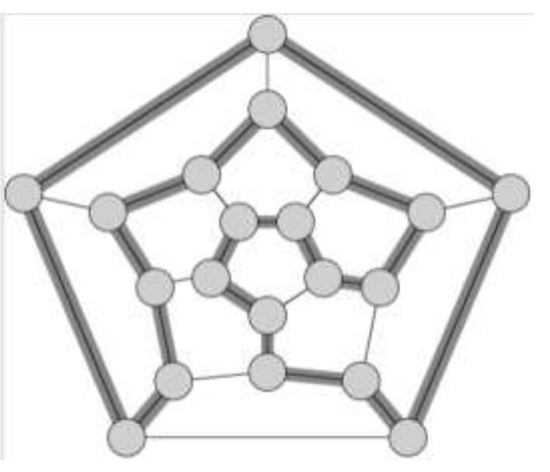
a) A graph representing the vertices, edges, and faces of a dodecahedron, with a hamiltonian cycle shown by shaded edges. In other words this graph is hamiltonian
b) A bipartite graph with an odd number of vertices. Any such graph is nonhamiltonian

Hamiltonian-cycle Problem

- "Does a graph G have a hamiltonian cycle?"

$$\mathrm{HAM-CYCLE} = \{\langle G \rangle : G \text{ is a hamiltonian graph}\}$$

- One possible solution:
  List all possible permutations of vertices of G and then check each permutation to see if it's a hamiltonian cycle or not
- Running time $\Omega(m!) = \Omega(\sqrt{n}) = \Omega\left(2^{\sqrt{n}}\right)$
- Where
  - $m =$ vertices of G
  - $n = |\langle G \rangle|$, that is the length of the encoded form of G
- Hamiltonian-cycle problem is actually NP-complete



Traveling Salesman problem

- Input:
  - Weighted graph G
  - Length ℓ
- Output:
  - Yes if a circuit exists of length ≤ ℓ
  - No otherwise
- TSP can be reduced from Hamiltonian cycle. TSP can be represented as a subroutine of HC, so as to represent TSP as NPC.

References

1) NP-Completeness - TUSHAR KUMAR J.
& RITESH BAGGA
2) Introduction to Algorithms (Cormen et al)
3) Wikipedia.com