CSE 373: Analysis of Algorithms

Selected Slides from Lecture 11
(Reductions and NP-Completeness)

Rezaul A. Chowdhury

Department of Computer Science
SUNY Stony Brook
Spring 2021

Optimization Problems vs Decision Problems

Optimization Problem:

An *optimization problem* is one in which each feasible (i.e., "legal") solution has an associated value, and we wish to find a feasible solution with the best value.

For example, in a problem that we call SHORTEST-PATH we are given a directed graph G and vertices u and v, and we wish to find a path from u to v that uses the fewest edges.

Decision Problem:

A *decision problem* is one in which the answer (i.e., solution) is simply Y/Yes (1) or N/No (0).

For example, in a problem that we call PATH we are given a directed graph G, vertices u and v, and an integer k, and we ask: does a path exist from u to v consisting of at most k edges?

P, NP, NP-Complete, NP-Hard

Complexity Class P:

The class P is the set of decision problems that are solvable in polynomial time.

More specifically, they are problems that can be solved in time $O(n^k)$ for some constant k, where n is the size of the input to the problem.

P, NP, NP-Complete, NP-Hard

Complexity Class NP:

The class NP is the set of decision problems with the following property: If the answer is Y, then there is a proof of this fact that can be checked in polynomial time.

Intuitively, NP is the set of decision problems where we can verify a Y answer quickly if we have the solution in front of us.

For example, in the HAMILTONIAN-CYCLE problem, given a directed graph $G = (V, E)$, a solution would be a sequence $\langle v_1, v_2, \dots, v_{|V|} \rangle$ of $|V|$ vertices. We could easily check in polynomial time that $(v_i, v_{i+1}) \in E$ for $i = 1,2,3, \dots, |V| - 1$ and that $\left(v_{|V|}, v_1\right) \in E$ as well.

Note that P ⊆ NP.

P, NP, NP-Complete, NP-Hard

NP-Complete Problems:

A problem is NP-complete if it is in NP and is as "hard" as any problem in NP.

If any NP-complete problem can be solved in polynomial time, then every problem in NP has a polynomial-time algorithm.

Hard problems (**NP**-complete)	Easy problems (in **P**)
3SAT	2SAT, HORN SAT
TRAVELING SALESMAN PROBLEM	MINIMUM SPANNING TREE
LONGEST PATH	SHORTEST PATH
3D MATCHING	BIPARTITE MATCHING
KNAPSACK	UNARY KNAPSACK
INDEPENDENT SET	INDEPENDENT SET on trees
INTEGER LINEAR PROGRAMMING	LINEAR PROGRAMMING
RUDRATA PATH	EULER PATH
BALANCED CUT	MINIMUM CUT

P, NP, NP-Complete, NP-Hard

NP-Hard Problems:

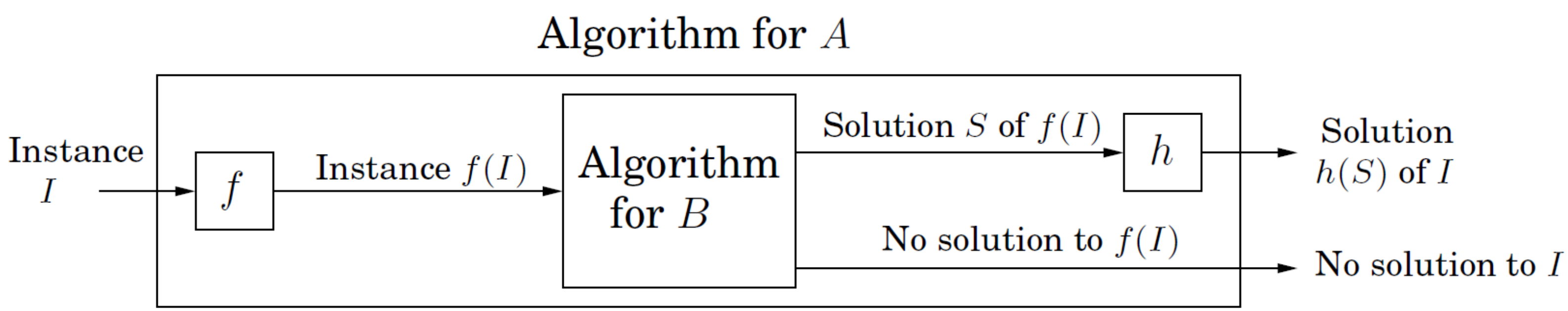
A problem is NP-hard if it is as “hard” as any problem in NP but may or may not belong to NP.

For example, optimization versions of NP-Complete decision problems are NP-Hard.

Reductions

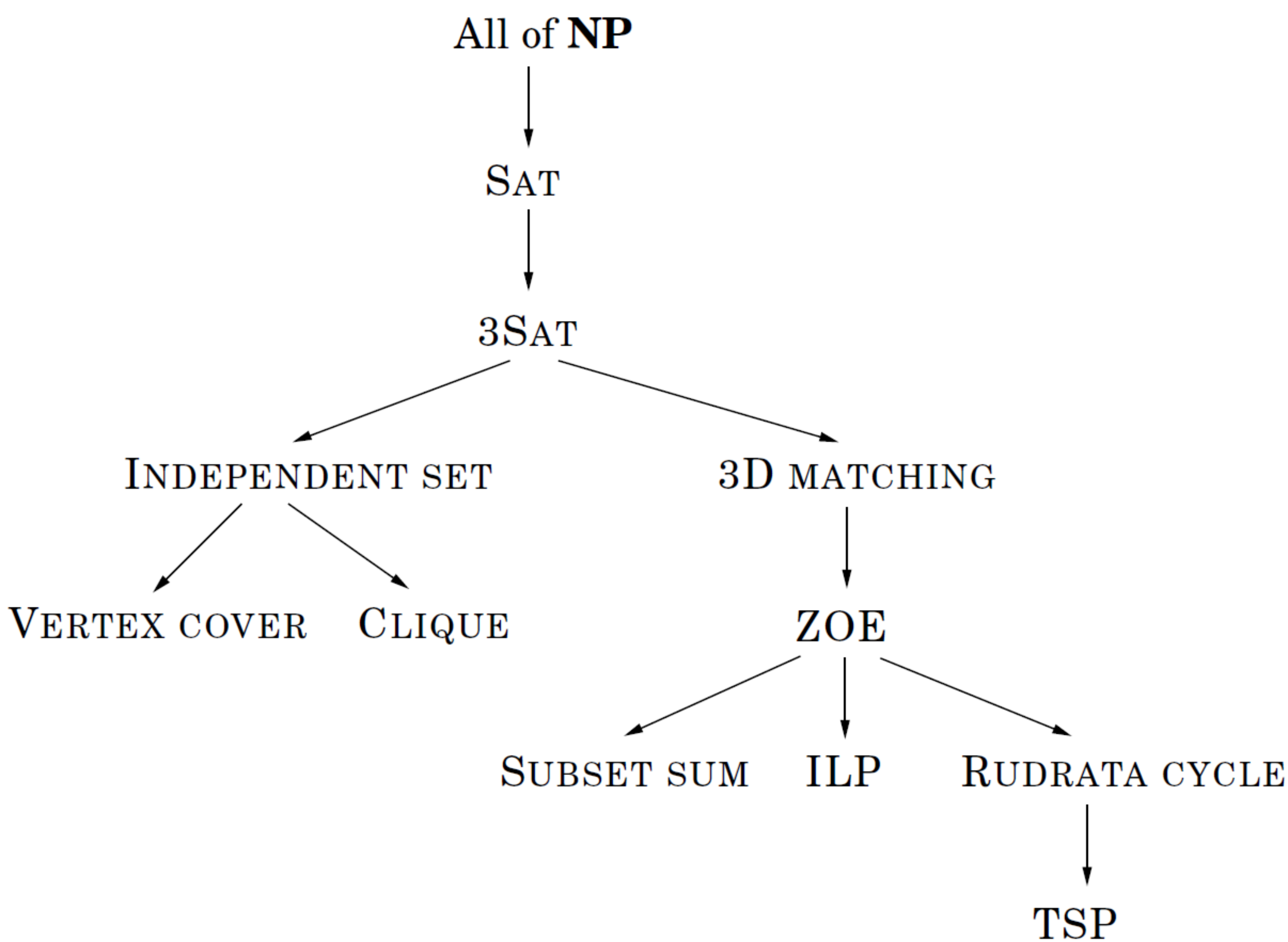
A reduction from a problem A to problem B is a polynomial-time algorithm f that transforms any instance I of A into an instance $f(I)$ of B, together with another polynomial-time algorithm h that maps any solution S of $f(I)$ back into a solution $h(S)$ of I.

If $f(I)$ has no solution, then neither does I.



A decision problem is NP-Complete if all other problems in NP can be reduced to it.

Reductions


All of NP
SAT
3SAT
INDEPENDENT SET
3D MATCHING
VERTEX COVER
CLIQUE
ZOE
SUBSET SUM
ILP
RUDRATA CYCLE
TSP

Decision Problem: SATISFIABILITY (SAT)

The SAT problem is the following: given a Boolean formula in conjunctive normal form, either find a satisfying truth assignment or else report that none exists.

$$(x \vee y \vee z) \wedge (x \vee \bar{y}) \wedge (y \vee \bar{z}) \wedge (z \vee \bar{x}) \wedge (\bar{x} \vee \bar{y} \vee \bar{z})$$

This is a Boolean formula in conjunctive normal form (CNF). It is a collection of clauses (the parentheses), each consisting of the disjunction (logical or, denoted $\vee$) of several literals, where a literal is either a Boolean variable (such as x) or the negation of one (such as $\bar{x}$).

A satisfying *truth* assignment is an assignment of *false* or *true* to each variable so that every clause contains a literal whose value is *true*.

Decision Problem: k-SATISFIABILITY (k-SAT)

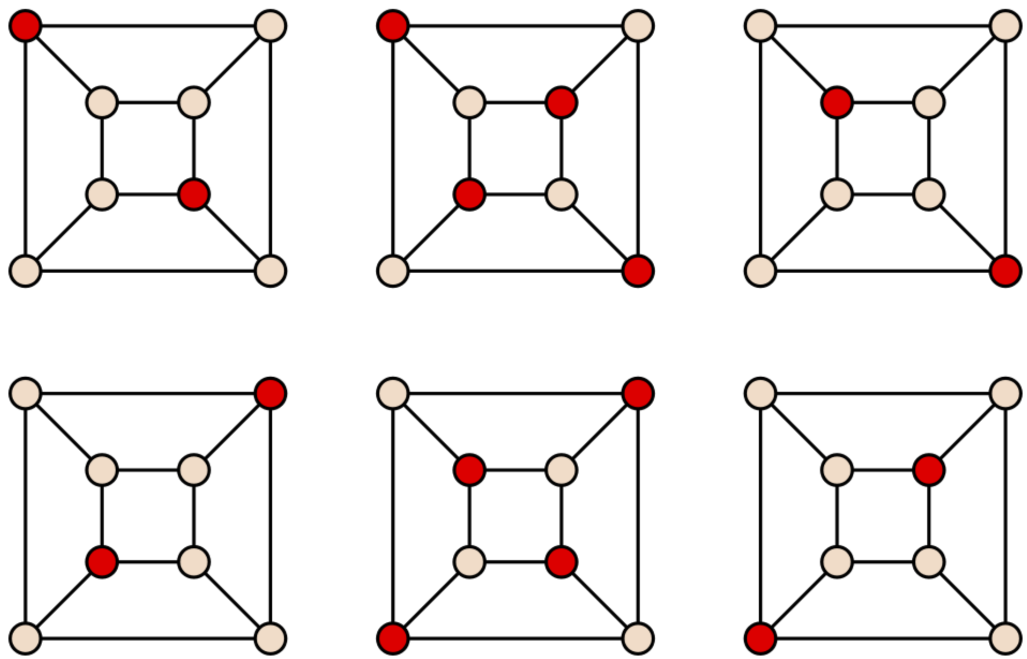
The k-SAT problem is a restricted version of the SAT problem, where each clause in the given Boolean formula is restricted to have at most k literals.

For example, the following has at most 3 literals per clause:

$$(x \vee y \vee z) \wedge (x \vee \bar{y}) \wedge (y \vee \bar{z}) \wedge (z \vee \bar{x}) \wedge (\bar{x} \vee \bar{y} \vee \bar{z})$$

Decision Problem: INDEPENDENT SET

In the INDEPENDENT SET problem, we are given a graph and an integer k, and the aim is to find k vertices that are independent, that is, no two of which have an edge between them.

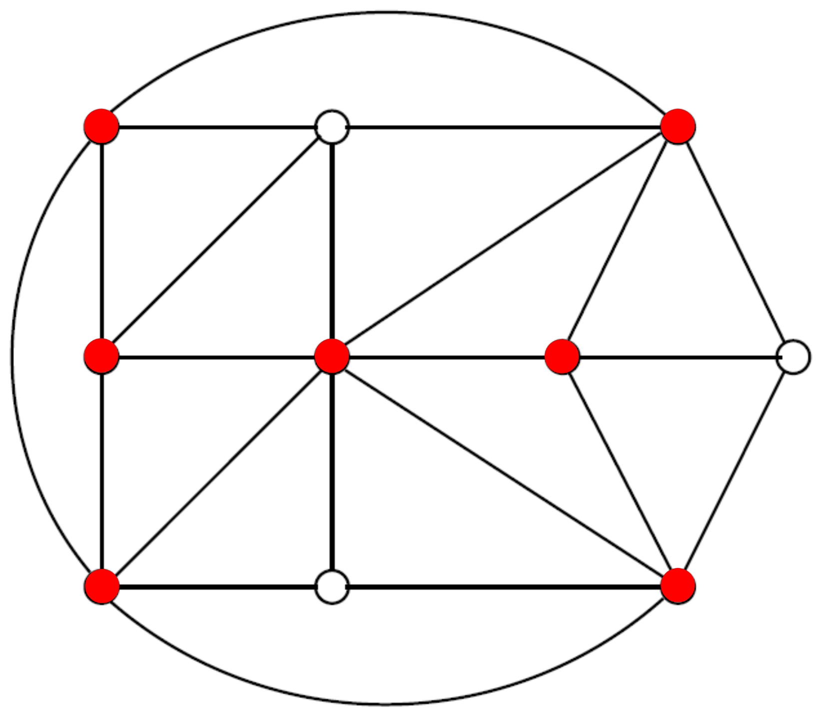


Some Independent Sets (red vertices) of the Cube Graph
Source: Wikipedia

Decision Problem: VERTEX COVER

In the VERTEX COVER problem, we are given a graph G and a budget k, and we are asked if there exists a set of k vertices that cover (touch) every edge.

All edges of the following graph can be covered with 7 vertices.

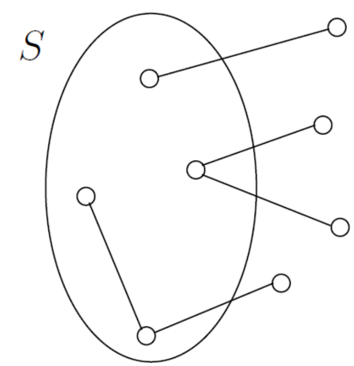


Can they be covered using only 6 vertices?

Reduction: INDEPENDENT SET → VERTEX COVER

To reduce INDEPENDENT SET to VERTEX COVER we just need to notice that a set of nodes S is a vertex cover of graph $G = (V, E)$ (that is, S touches every edge in E) if and only if the remaining nodes, $V \setminus S$ are an independent set of G.

S is a vertex cover if and only if $V - S$ is an independent set.

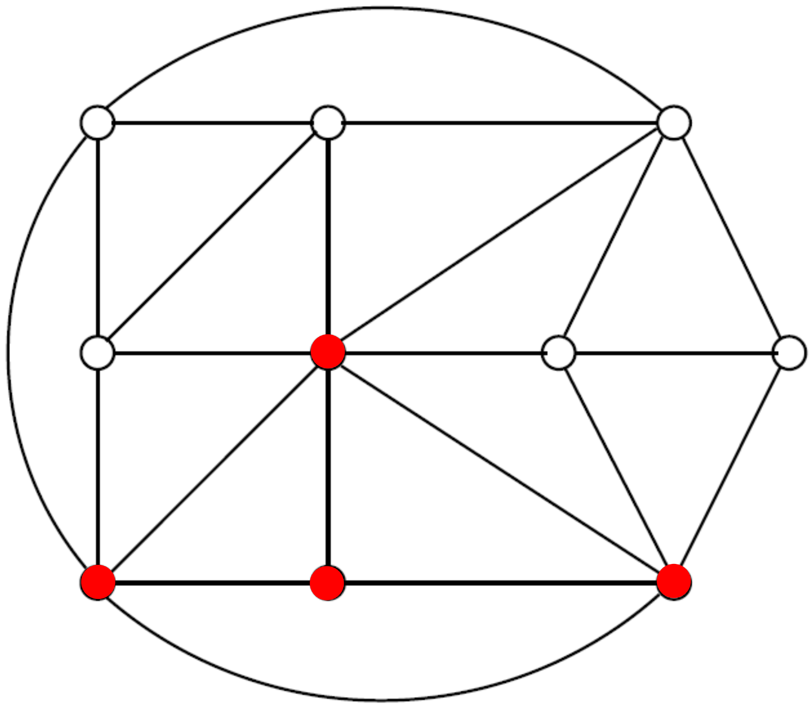


Therefore, to solve an instance (G, k) of INDEPENDENT SET, simply look for a vertex cover of G with $|V| - k$ nodes. If such a vertex cover exists, then take all nodes not in it. If no such vertex cover exists, then G cannot possibly have an independent set of size k.

Decision Problem: CLIQUE

In the CLIQUE problem, we are given a graph G and a positive integer k, and we are asked if there exists a set of k vertices such that all edges between them are present.

There is a clique of size 4 in the following graph.



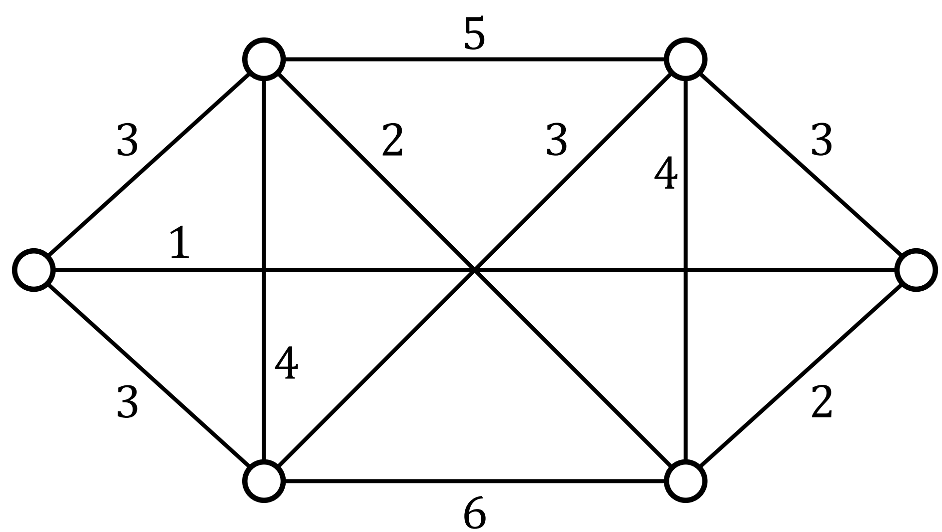
What is the size of the largest clique in the graph above?

Decision Problem: TRAVELING SALESMAN PROBLEM (TSP)

In the TRAVELING SALESMAN PROBLEM (TSP) we are given n vertices $1,2,\dots,n$ and all $n(n-1)/2$ distances between them, as well as a budget b. We are asked to find a *tour*, a cycle that passes through every vertex exactly once, of total cost b or less, or to report that no such tour exists.

We seek a permutation $\tau(1),\tau(2),\dots,\tau(n)$ of the vertices such that when they are toured in this order, the total distance covered $\le b$:

$$d_{\tau(1),\tau(2)} + d_{\tau(2),\tau(3)} + \cdots + d_{\tau(n-1),\tau(n)} + d_{\tau(n),\tau(1)} \le b.$$

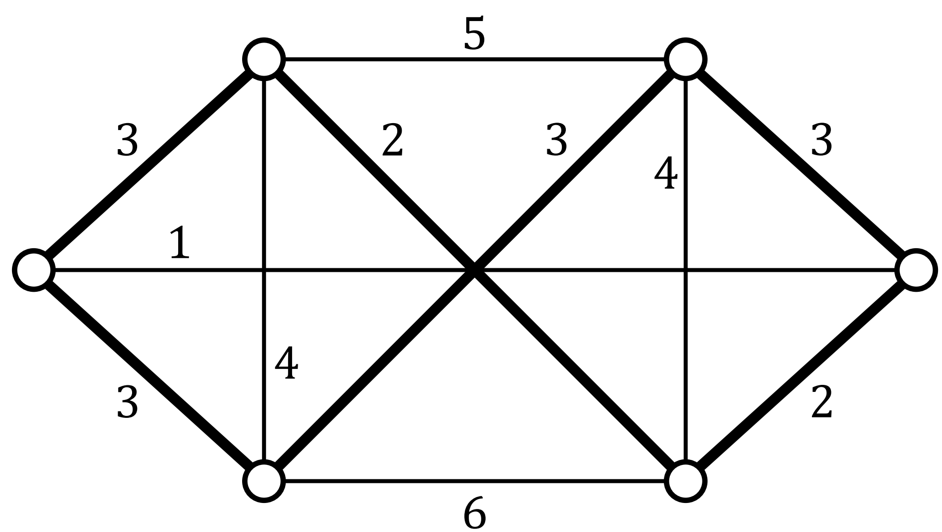


Decision Problem: Traveling Salesman Problem (TSP)

In the Traveling Salesman Problem (TSP) we are given n vertices $1,2,\dots,n$ and all $n(n-1)/2$ distances between them, as well as a budget b. We are asked to find a *tour*, a cycle that passes through every vertex exactly once, of total cost b or less, or to report that no such tour exists.

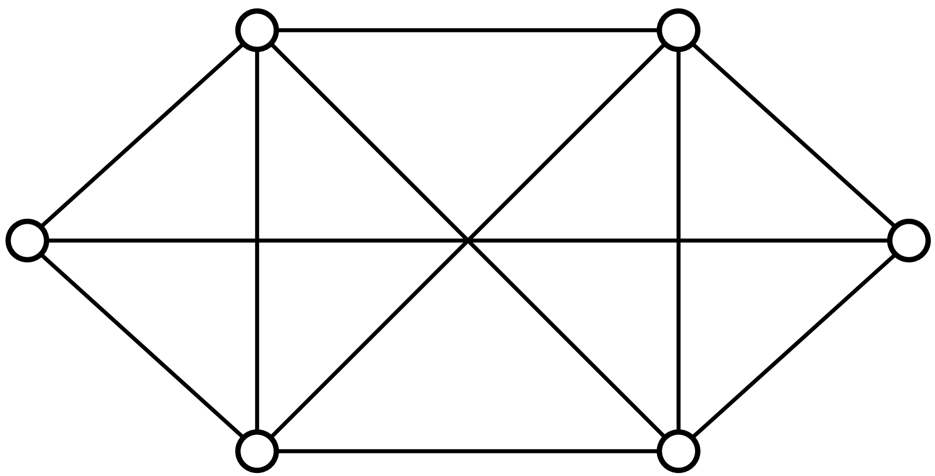
We seek a permutation $\tau(1),\tau(2),\dots,\tau(n)$ of the vertices such that when they are toured in this order, the total distance covered $\le b$:

$$d_{\tau(1),\tau(2)} + d_{\tau(2),\tau(3)} + \cdots + d_{\tau(n-1),\tau(n)} + d_{\tau(n),\tau(1)} \le b.$$



Decision Problem: HAMILTONIAN CYCLE

In the HAMILTONIAN CYCLE problem, we are given a graph $G = (V, E)$, and we are asked to a find a cycle that visits each vertex in V exactly once, or report that no such cycle exists.



Decision Problem: HAMILTONIAN CYCLE

In the HAMILTONIAN CYCLE problem, we are given a graph $G = (V, E)$, and we are asked to a find a cycle that visits each vertex in V exactly once, or report that no such cycle exists.