UNIT - 5

NP and Computational Intractability

UNIT - 5

NP and Computational Intractability: Polynomial-Time Reductions NP-Complete Problems: Circuit Satisfiability: A First NP-Complete Problem, General Strategy for Proving New Problems NP Complete, Sequencing Problems: The Traveling Salesman Problem, The Hamiltonian Cycle Problem. **(Text book - 1, 8.1, 8.4)**

(Text book - 2, 12.1,12.2)

P Class

The P in the P class stands for **Polynomial Time.** It is the collection of decision problems(problems with a "yes" or "no" answer) that can be solved by a deterministic machine in polynomial time

NP Class

The NP in NP class stands for **Non-deterministic Polynomial Time**. It is the collection of decision problems that can be solved by a non-deterministic machine (note that our computers are deterministic) in polynomial time.

NP-hard class

An NP-hard problem is at least as hard as the hardest problem in NP and it is a class of problems such that every problem in NP reduces to NP-hard.

NP-complete class

A problem is NP-complete if it is both NP and NP-hard. NP-complete problems are the hard problems in NP. A decision problem is NP-complete if it is in **NP** and every problem in NP is reducible to it in polynomial time

Complexity Class	Characteristic feature
P	Easily solvable in polynomial time.
NP	Yes, answers can be checked in polynomial time.
Co-NP	No, answers can be checked in polynomial time.
NP-hard	All NP-hard problems are not in NP and it takes a long time to check them.
NP-complete	A problem that is NP and NP-hard is NP-complete.

NP-hard	NP-Complete
Polynomial time verification by a deterministic machine is not necessary.	Can be verified in Polynomial time by a deterministic machine.
NP-hard is not a decision problem.	NP-Complete is exclusively a decision problem.
Not all NP-hard problems are NP-complete.	All NP-complete problems are NP-hard
Do not have to be a Decision problem.	It is exclusively a Decision problem.
Need not to be a NP problem	Must be NP
Example: Halting problem, Vertex cover problem, etc.	Example: Determine whether a graph has a Hamiltonian cycle, Determine whether a Boolean formula is satisfiable or not, Circuit-satisfiability problem and problems that are NP hard.

Computational Intractability

A problem is computationally intractable if there is no known algorithm that can solve all instances of the problem in polynomial time (i.e., efficiently) as the input size grows.

Relationship Between NP and Intractability

- All NP-complete and NP-hard problems are considered intractable because they cannot be solved in polynomial time.
- Intractability highlights the limits of algorithmic efficiency and motivates the study of approximation algorithms, heuristics, and special cases.

Polynomial-Time Reductions NP-Complete Problems

- Problems that can be solved in polynomial time are called **tractable,** and problems that cannot be solved in polynomial time are called **intractable**
- **Class P** is a class of decision problems that can be solved in polynomial time by **(deterministic)** algorithms. This class of problems is called polynomial.
- **Class NP** is the class of decision problems that can be solved by **nondeterministic** polynomial algorithms. This class of problems is called nondeterministic polynomial.

A problem **A** is **polynomial-time reducible** to a problem **B**, if there exists a polynomial-time algorithm that transforms instances of A into instances of B, such that answer to A is "yes" **if** answer to B is "yes". This concept allows us to **compare the hardness** of problems.

Proving NP-Completeness (Common Approach):

- Step 1: Show the problem is in NP.
- Step 2: Choose a known NP-Complete problem.

- Step 3: Reduce the known problem to your target problem in polynomial time.

Implication of NP-Completeness:

- If any NP-Complete problem can be solved in polynomial time, **then all problems in NP can be solved in polynomial time** — i.e., **P = NP**.
- No polynomial-time algorithm is currently known for any NP-Complete problem.

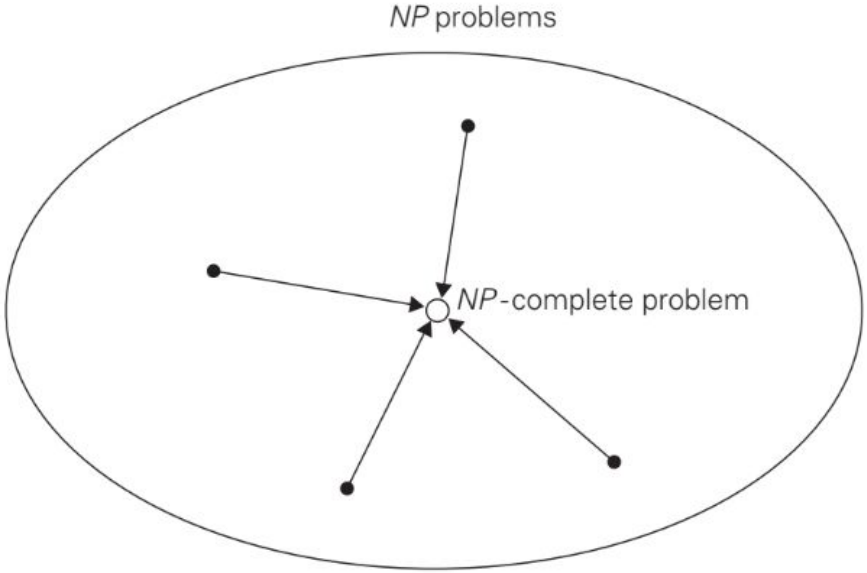


FIGURE 11.6 Notion of an *NP*-complete problem. Polynomial-time reductions of *NP* problems to an *NP*-complete problem are shown by arrows.

For more Info:

https://www.youtube.com/watch?v=valpD5Jx7aw&t=5s&pp=ygUzQ2lyY3VpdCBTYXRpc2ZpYWJpbGl0eTogQSBGaXJzdCBOUC1Db21wbGV0ZSBQcm9ibGVt

Circuit Satisfiability: A First NP-Complete Problem

- **The problem asks:** Given a Boolean circuit made up of logic gates (AND, OR, NOT) and a single output, is there an assignment of inputs that makes the output TRUE? It's a **decision problem** — the answer is either YES (satisfiable) or NO (not satisfiable)
- **Circuit-SAT** was the **first problem proven to be NP-complete**, making it a cornerstone in the theory of NP-completeness.

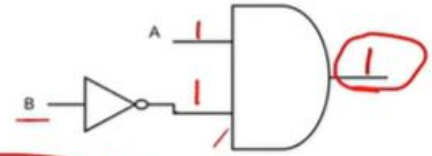
The satisfiability problem

Given a Boolean combinational circuit composed of AND, OR, and NOT gates, is it satisfiable?

A circuit is said to be satisfiable if a truth assignment that causes the output of the circuit to be 1

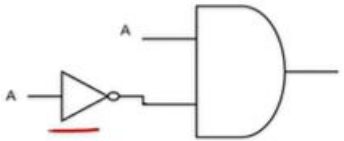
if the circuit has k inputs, then we would have to check up to 2^k possible assignments

NP



A B
0 0
1 0 1

A=1 B=0



Cooks Proved that Satisfiability is an NP complete Problem

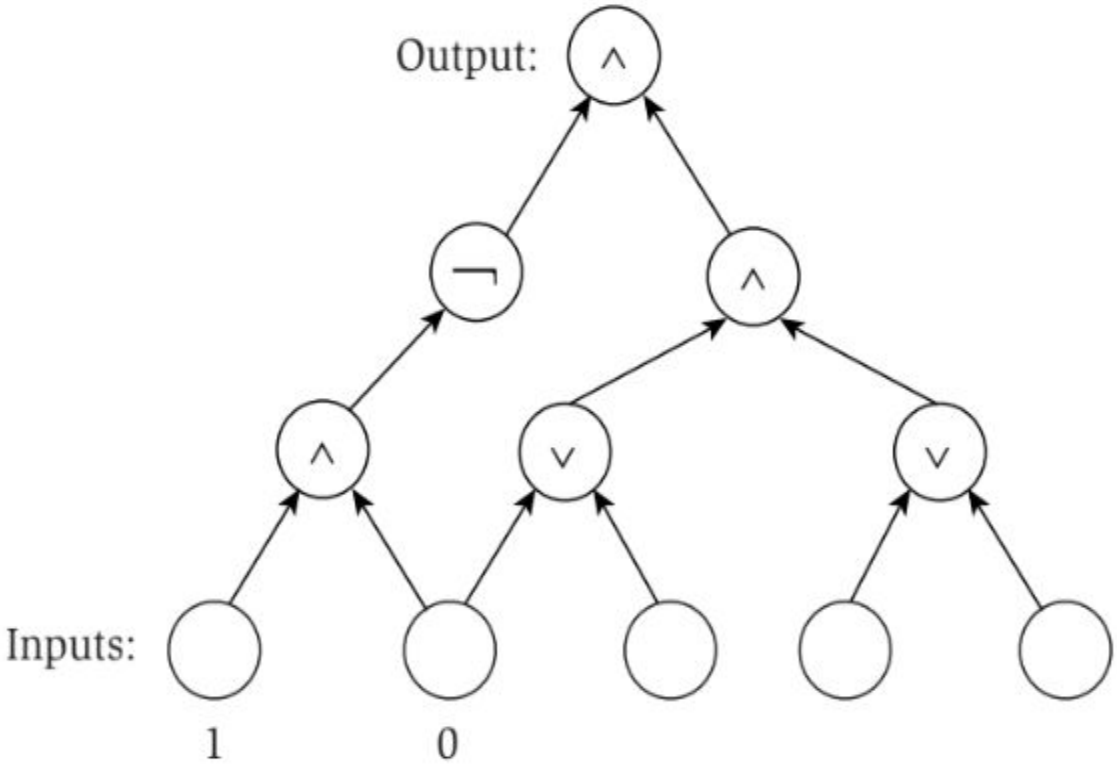


Figure 8.4 A circuit with three inputs, two additional sources that have assigned truth values, and one output.

NP-Completeness of SAT:

We have to show that if C-SAT is NP-Complete than SAT is also NP-Complete.

A circuit C is satisfiable iff the formula f(C) is Satisfiable.

If C is satisfiable, then there exists an assignment to it's input wires that results in the output being 1.

Therefore there exists an assignment to all intermediate i/p o/p wires, resulting in a satisfying assignment to the formula f(C).

If f(C) is satisfiable, then any satisfying assignment results in a satisfying assignment to the circuit C, by restricting it to variable for the input wires.

NP-Completeness of SAT:

The reduction function f can be computed in polynomial time .

This can be done by performing a breadth-first search on the graph representation of the circuit generating a clause for each gate.

Hence SAT is NP-Complete.

General Strategy for Proving New Problems NP Complete

Since it is widely believed that **P = NP**, the discovery that a problem is NP-complete can be taken as a strong indication that it cannot be solved in polynomial time.

Given a new problem X, here is the basic strategy for proving it is NP-complete.

1. Prove that $X \in NP$.
2. Choose a problem Y that is known to be NP-complete.
3. Prove that $Y \leq_P X$.

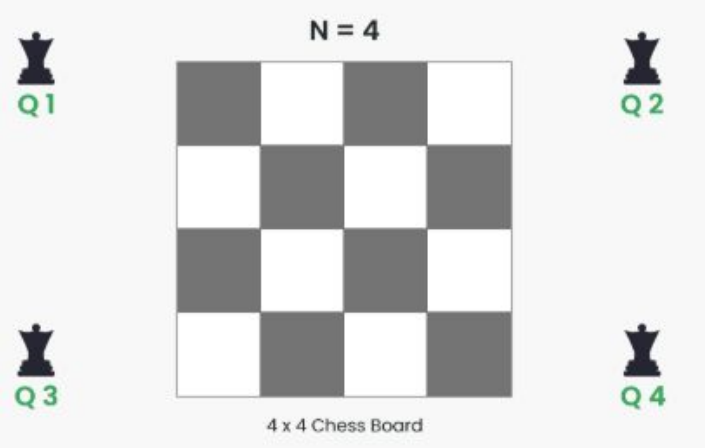
When we use this style of reduction, we can refine strategy above to the following outline of an NP-completeness proof.

1. Prove that X $\in$ NP.
2. Choose a problem Y that is known to be NP-complete.
3. Consider an arbitrary instance sY of problem Y, and show how to construct, in polynomial time, an instance sX of problem X that satisfies the following properties:

- If sY is “yes” instance of Y, then sX is “yes” instance of X.
- If sX is “yes” instance of X, then sY is “yes” instance of Y.

n-Queens Problem (backtracking technique)

The problem is to place n queens on an **n × n** chessboard so that **no two queens attack each other** by being in the same row or in the same column or on the same diagonal.



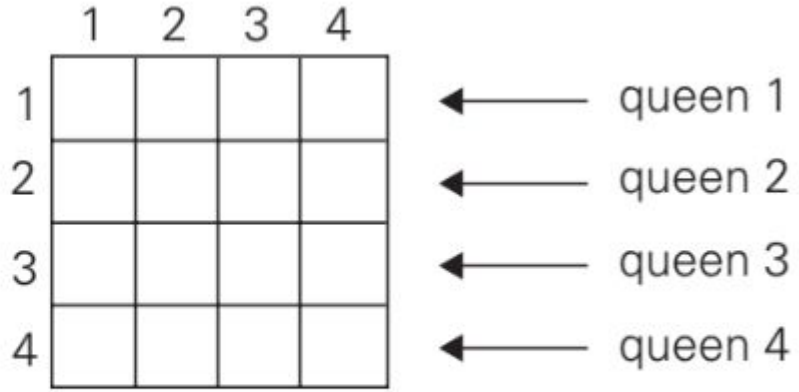


FIGURE 12.1 Board for the four-queens problem.

Using Backtracking

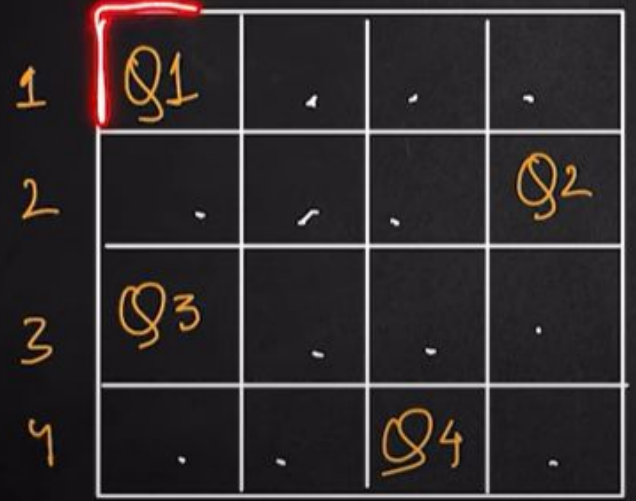
The idea is to use backtracking to check all possible combinations of n queens in a chessboard of order **n*n**.

- **Start** from the **first** row and for **each** row **place** queen at **different** columns and **check** for **clashes** with other queens.
- To check for clashes, iterate through all the rows of current column and both the diagonals. If it is **safe** to place queen in current column, **mark** the cell **occupied**.
- If at any row, there is **no safe** column to place the queen, **backtrack** to **previous** row and place the queen in other safe column and again check for the next row.

N Queens

n×n board → n Queens

n = 4



n Queens

① nQueens ⇒ n Rows

② Queen → safe

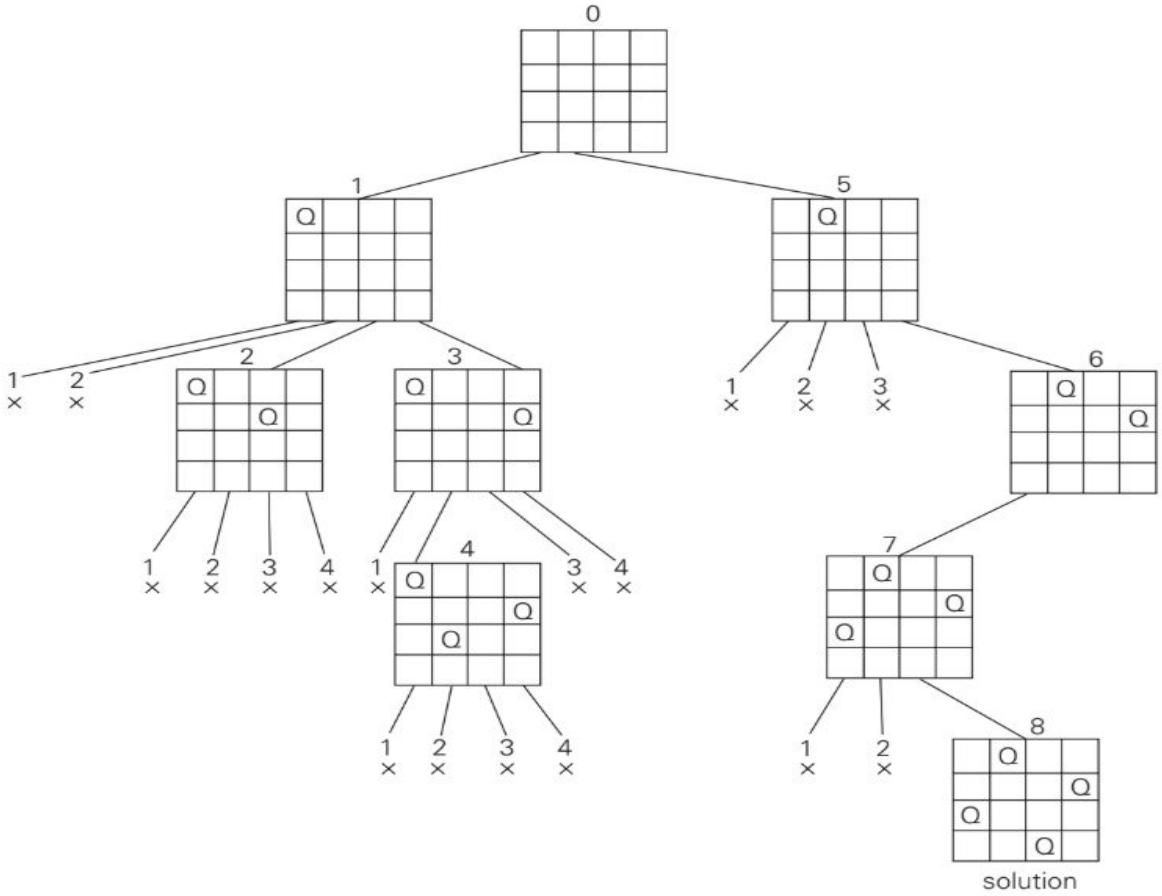


FIGURE 12.2 State-space tree of solving the four-queens problem by backtracking. × denotes an unsuccessful attempt to place a queen in the indicated column. The numbers above the nodes indicate the order in which the nodes are generated.

Travelling Salesman Problem

TRAVELLING SALESMAN PROBLEM - BRANCH & BOUND

Problem

Given n cities, a salesman starts at a specified city(often called source) visit all n-1 cities only once and return to the city from where he has started

Objective

- Find a route through the cities that minimize the cost thereby maximize the profit

Model

The vertices of the graphs represent the various cities

- The weights associated with edges represent the distances between two cities or the cost involved from one city to other city during travelling

Observations in Constructing State Space Tree

- Tour always starts at a
- In the tour the first and last city remains same whereas other intermediate nodes be distinct

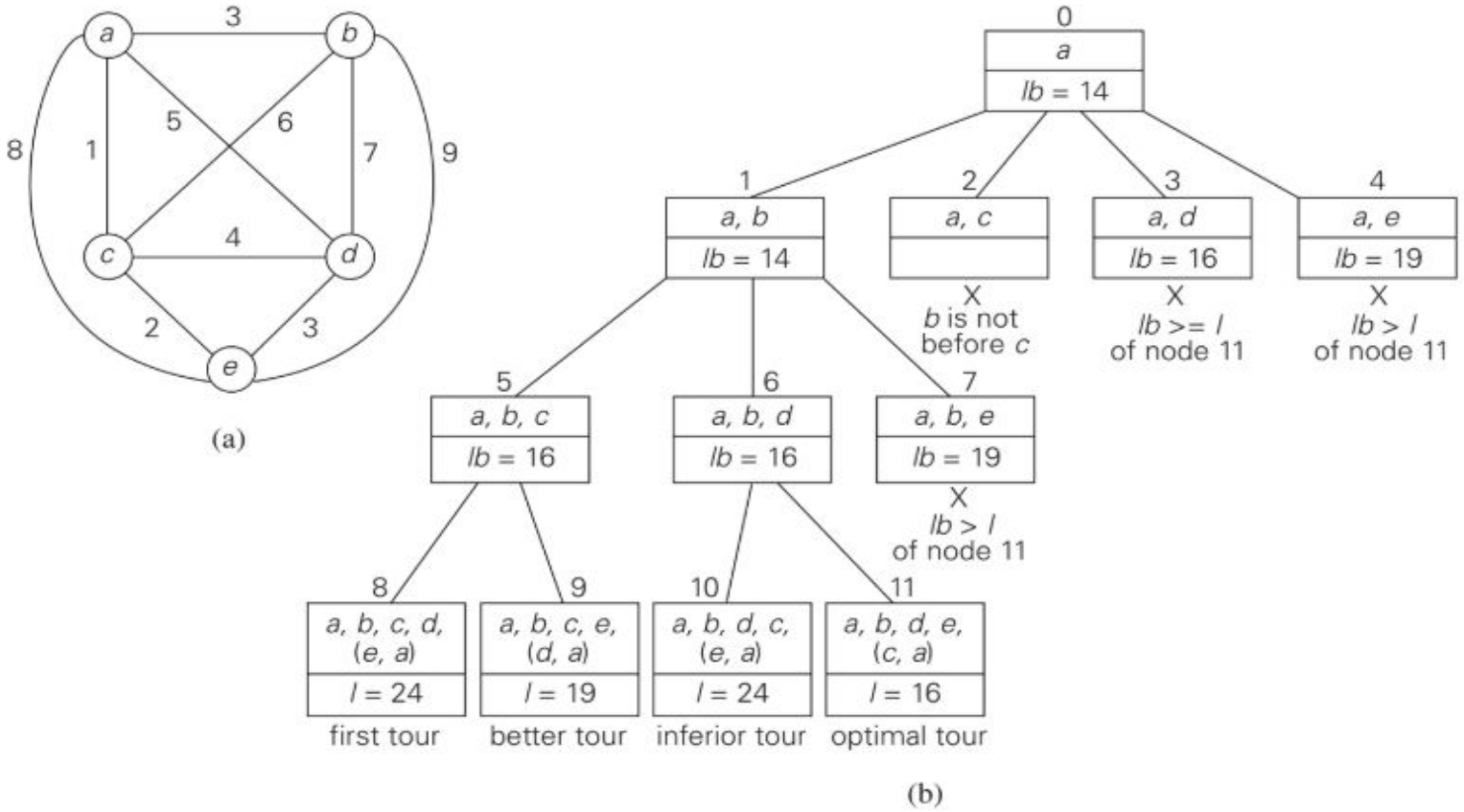


FIGURE 12.9 (a) Weighted graph. (b) State-space tree of the branch-and-bound algorithm to find a shortest Hamiltonian circuit in this graph. The list of vertices in a node specifies a beginning part of the Hamiltonian circuits represented by the node.

Practice Problem

Apply the branch-and-bound algorithm to solve the traveling salesman problem for the graph given in Fig. 9(b) considering 'a' source vertex:

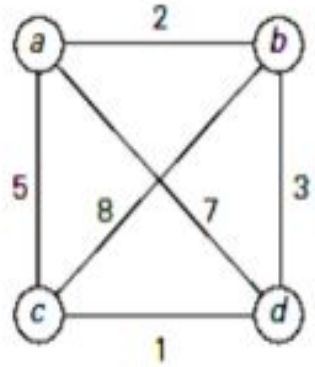


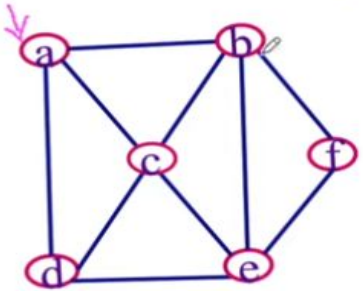
Fig. 9(b)

For More Info:https://youtu.be/-dbV4j8PkwQ?si=pKYYo_w5hYxGAaXc

Hamiltonian Cycle Problem

BACKTRACKING - HAMILTONIAN CIRCUIT

- Given a connected graph aim is to find whether a Hamiltonian Cycle is present or not
- Hamiltonian path is a path in an undirected graph that visits each vertex exactly once
- Hamiltonian circuit is a cycle in an undirected graph that visits each vertex exactly once and returns to the starting vertex
- TSP aim is to find the minimum cost of the tour
- Hamiltonian cycle aims to visit all vertices exactly only once and return back to starting vertex



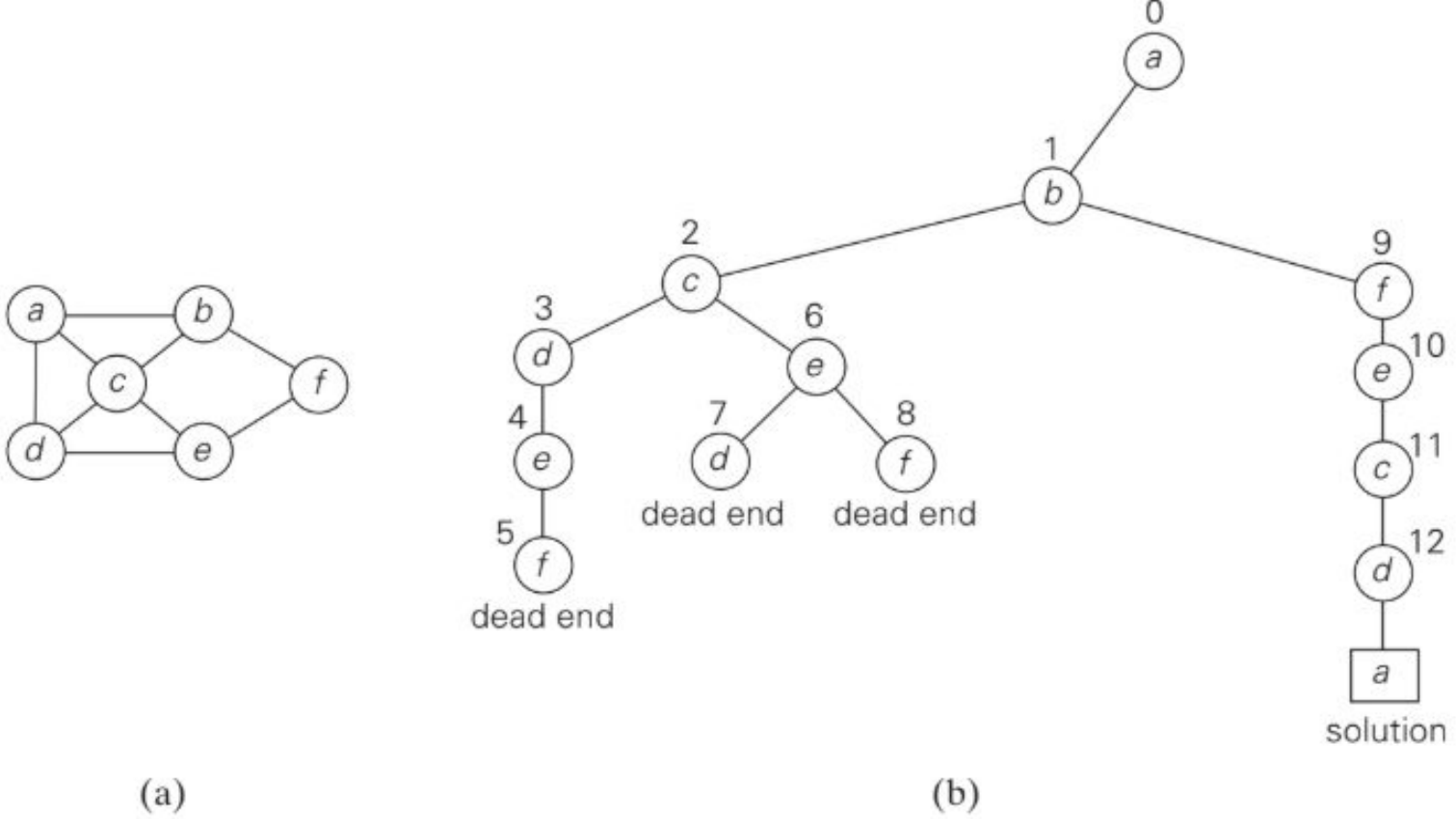


FIGURE 12.3 (a) Graph. (b) State-space tree for finding a Hamiltonian circuit. The numbers above the nodes of the tree indicate the order in which the nodes are generated.

Practice problem

Apply backtracking to the given problem of finding a Hamiltonian circuit in the graph given in Fig. 10(a).

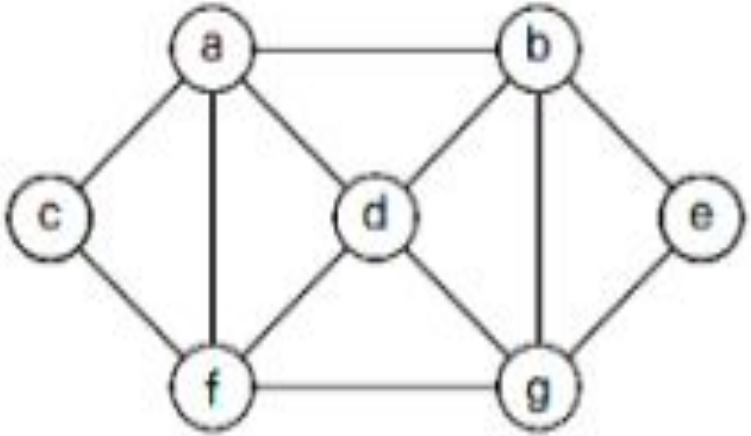


Fig. 10(a)

Describe Hamiltonian cycle. Find a Hamiltonian cycle in the graph given in Fig. 10(b).

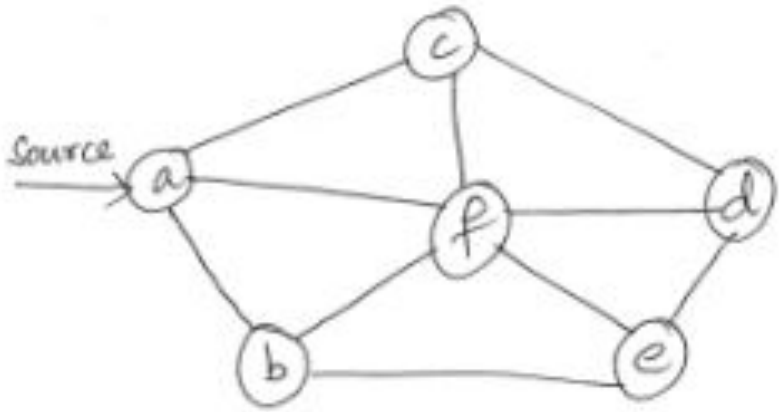


Fig 10(b)

For More Info: https://youtu.be/u9zNwN6z-q4?si=TYzWIuNZtV6YLXry