Mathematical Foundations of Computer Science

This Lecture: Graph Theory: Hamiltonian Graphs

Department of Mathematical and Computational Sciences
National Institute of Technology Karnataka, Surathkal

MA714 (Odd Semester [2025-26])

What is a Hamiltonian cycle?

A **Hamiltonian path** in a graph visits every vertex exactly once.

A **Hamiltonian cycle** (or circuit) is a Hamiltonian path that starts and ends at the same vertex.

A graph that contains a Hamiltonian cycle is called a **Hamiltonian graph**.

These are vertex-covering cycles and are fundamental in optimization problems (e.g., TSP).

Eulerian vs Hamiltonian: brief comparison

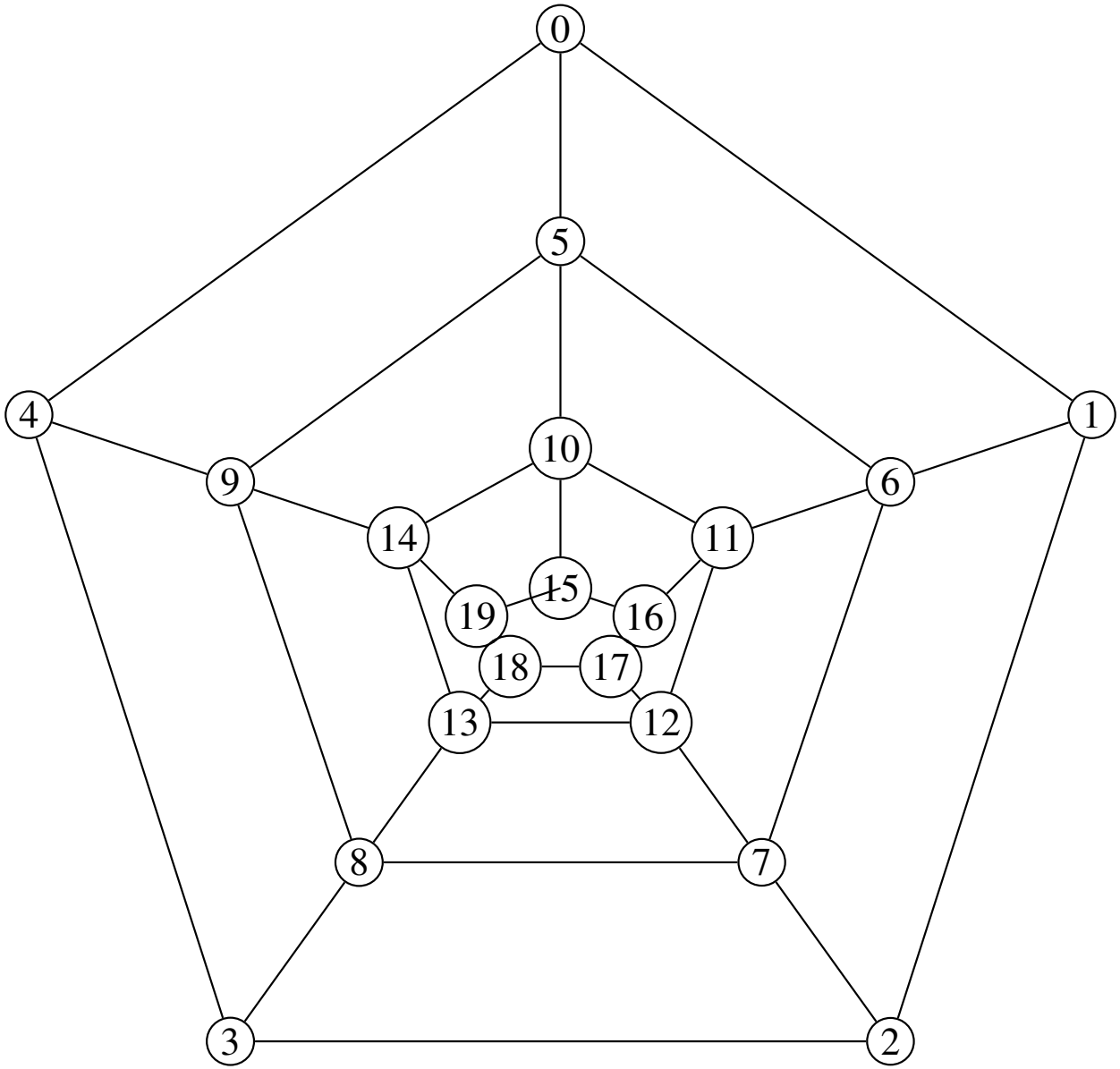
Eulerian: exists when there is a trail visiting every *edge* exactly once.

Existence has simple degree conditions (e.g., all vertices even degree for an Eulerian circuit).

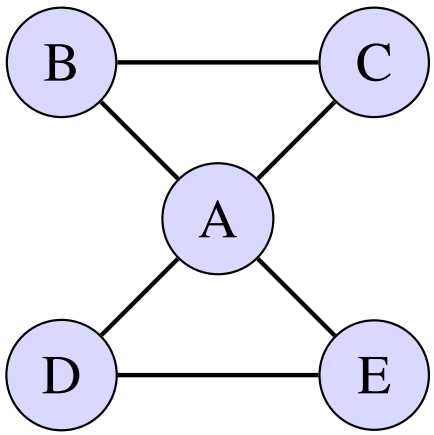
Hamiltonian: a cycle visiting every *vertex* exactly once.

Decision problem is **NP-complete**; no easy local degree condition.

Use table: Eulerian $\leftrightarrow$ edges; Hamiltonian $\leftrightarrow$ vertices.


0
5
4
1
10
9
6
14
11
15
19
16
18
17
13
12
8
7
3
2

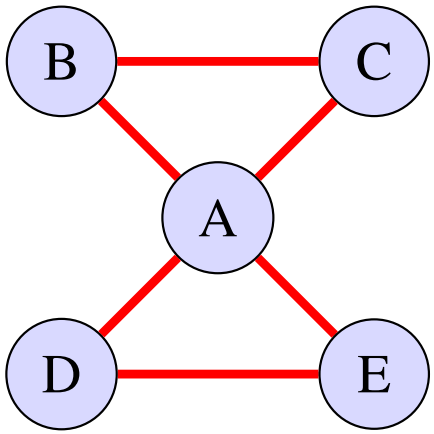
Eulerian vs Hamiltonian Graphs: Illustration



Eulerian Example: Bow-tie graph

All vertices even degree

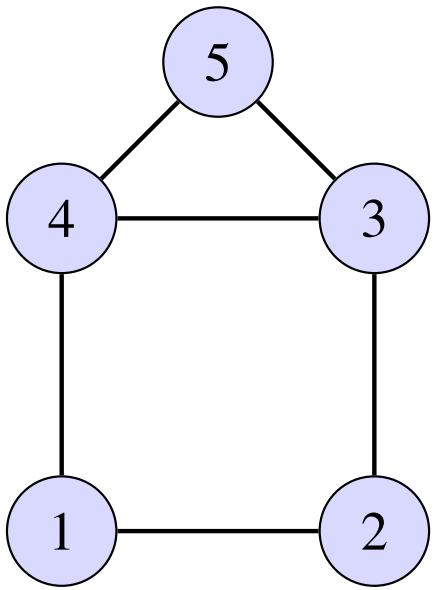
Eulerian vs Hamiltonian Graphs: Illustration



Eulerian Example: Bow-tie graph

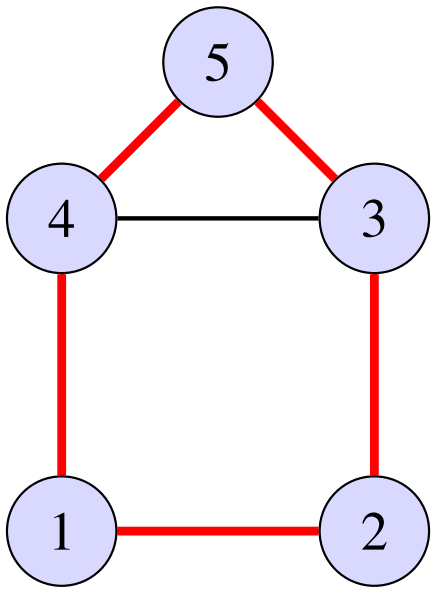
All vertices even degree

Eulerian vs Hamiltonian Graphs: Illustration



Hamiltonian Example: House graph
Has Hamiltonian cycle

Eulerian vs Hamiltonian Graphs: Illustration



Hamiltonian Example: House graph
Has Hamiltonian cycle

Traveling Salesman Problem (TSP) — searching for a minimum-weight Hamiltonian cycle.

Routing & Logistics — visiting nodes once (last-mile planning, vehicle routing).

Scheduling and timetabling — visiting tasks/slots exactly once.

Bioinformatics — reconstruction problems (Hamiltonian paths in overlap graphs).

Robotics / Coverage planning — sequences of distinct waypoints.

Backtracking / DFS search — exact but exponential, good for small graphs or heavy pruning.

Dynamic Programming (Held–Karp) — exact method, $O(n^2 2^n)$ time, exponential but better than brute force.

Heuristics / Approximations — nearest neighbor, Christofides (for metric TSP), local search (useful for large instances).

Sufficient conditions — Dirac's and Ore's theorems (useful to prove Hamiltonicity in special graphs).

Idea

Start at a chosen vertex and grow a path by repeatedly adding unvisited adjacent vertices. If you reach a vertex from which no valid extension exists, undo (backtrack) the last choice and try the next possibility. Continue until a Hamiltonian cycle is found or all choices are exhausted.

Key pruning checks:

- Only add vertices adjacent to the current last vertex.
- Avoid revisiting vertices already in the path.
- Optionally: degree/feasibility checks to prune early.

Pseudocode

1. Let path[0] = start vertex.
2. `function Backtrack(path, pos):`

   If `pos == n` and `edge(path[n-1], path[0])` exists $\Rightarrow$ **cycle found**.

   For each vertex v not in path:

   If edge(path[pos-1], v) exists:

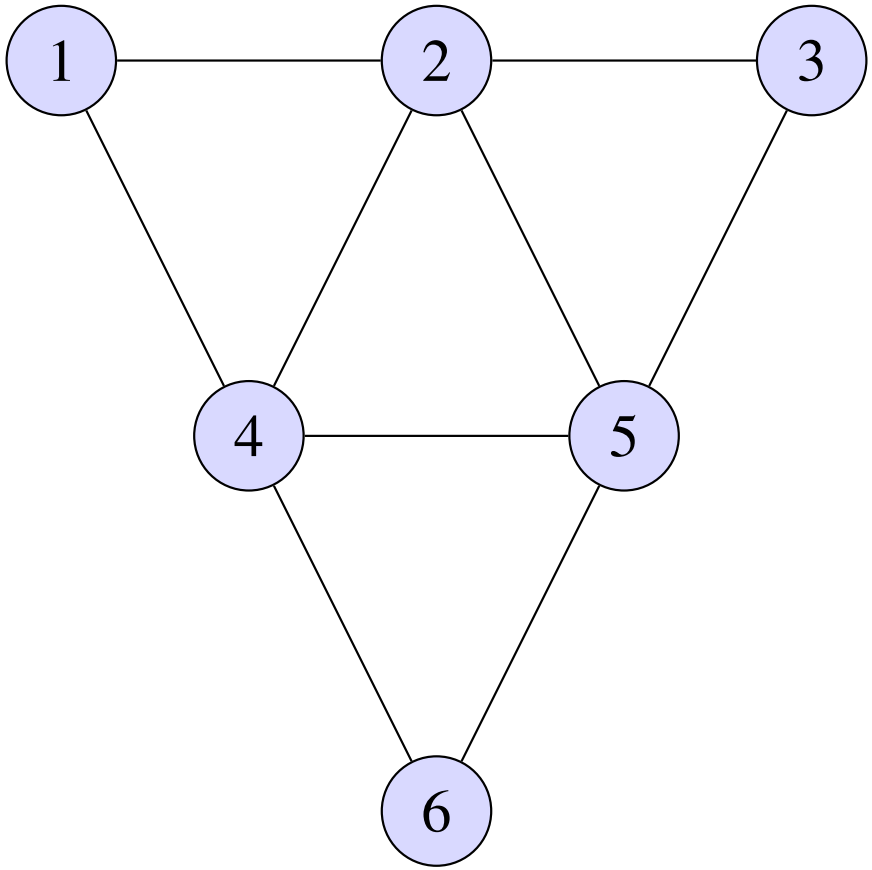
   path[pos] $\leftarrow$ v.

   If Backtrack(path, pos+1) returns true $\Rightarrow$ return true.

   Else remove v (backtrack).

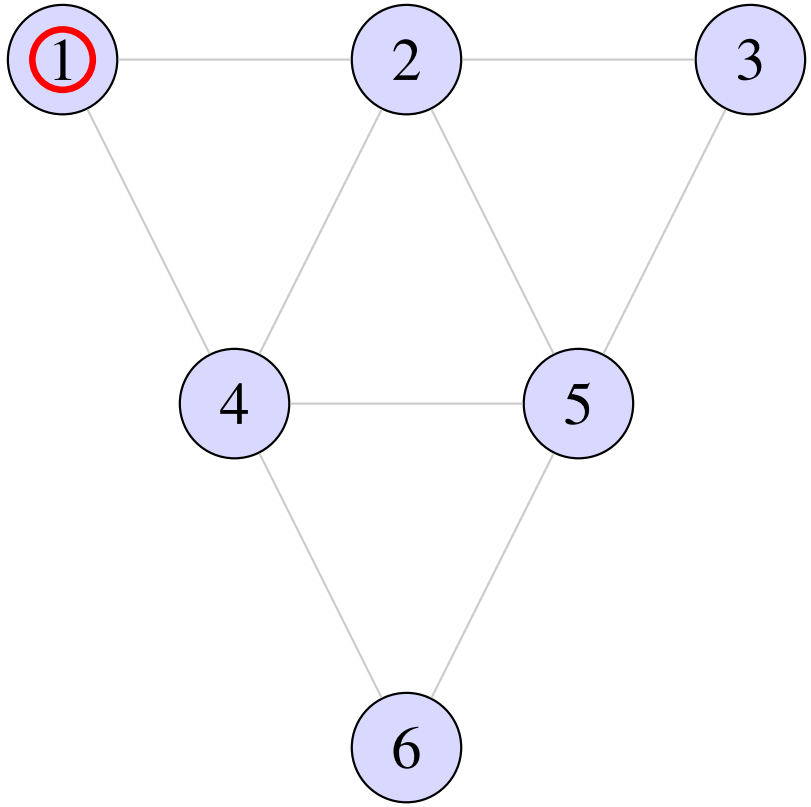
   Return false (no extension leads to a solution).

Example graph (6 vertices)



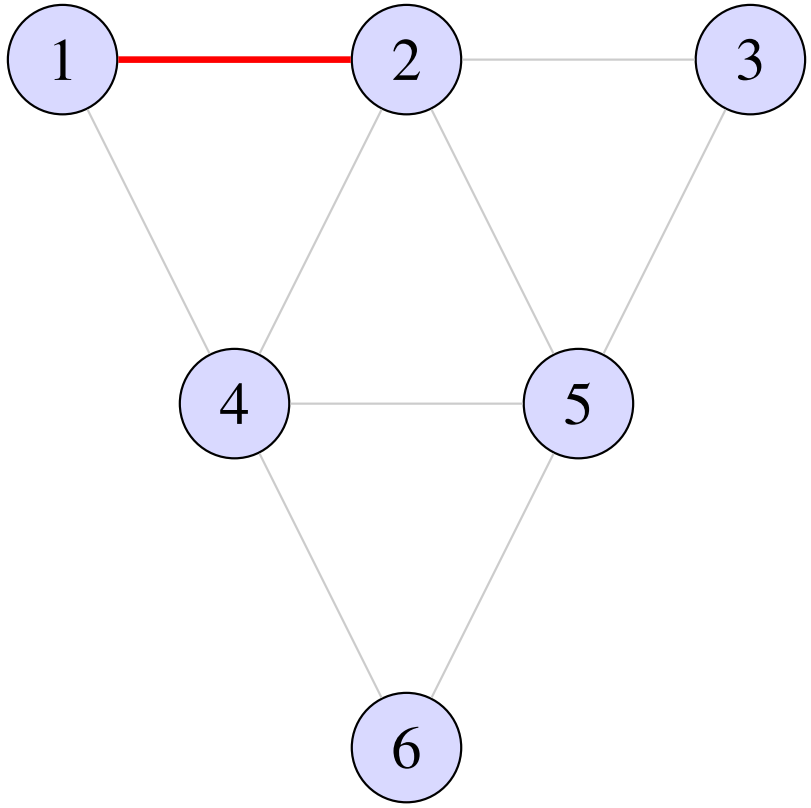
This example will be used for the stepwise backtracking demonstration.

Backtracking demonstration: step-by-step



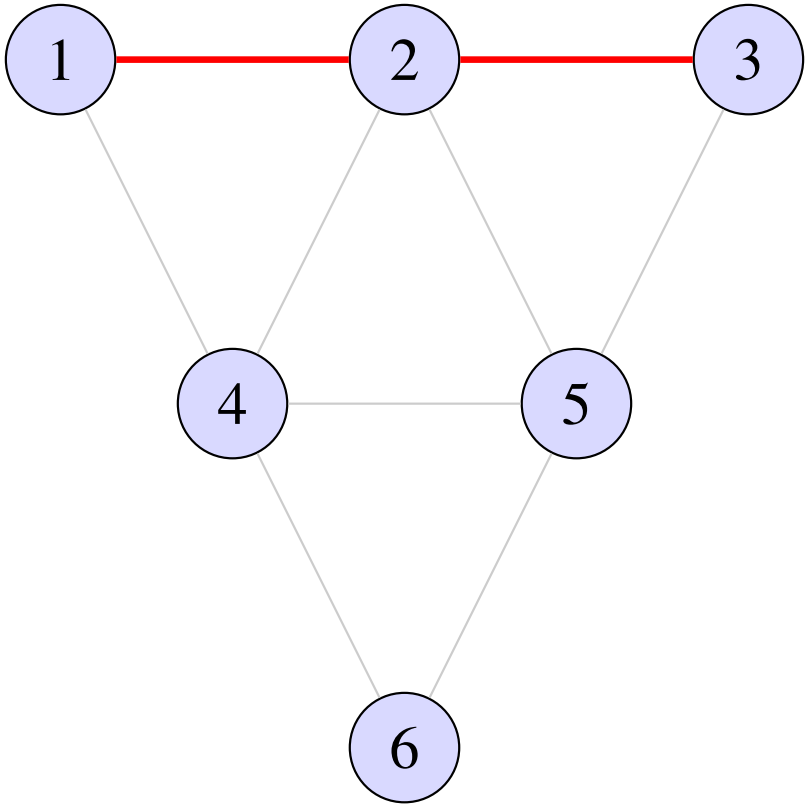
Path: (1)

Backtracking demonstration: step-by-step



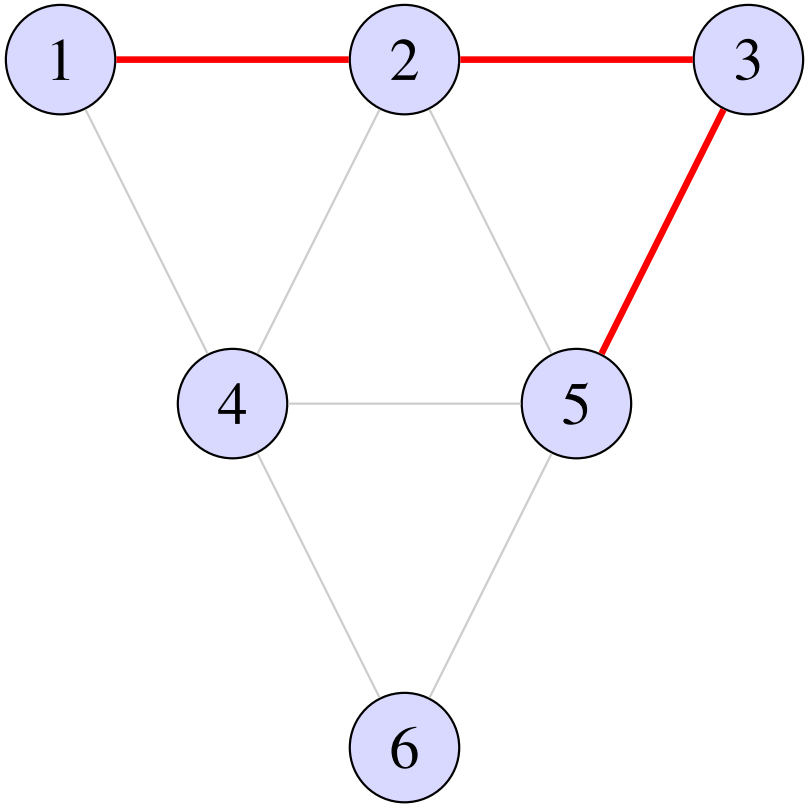
Path: $(1, 2)$

Backtracking demonstration: step-by-step



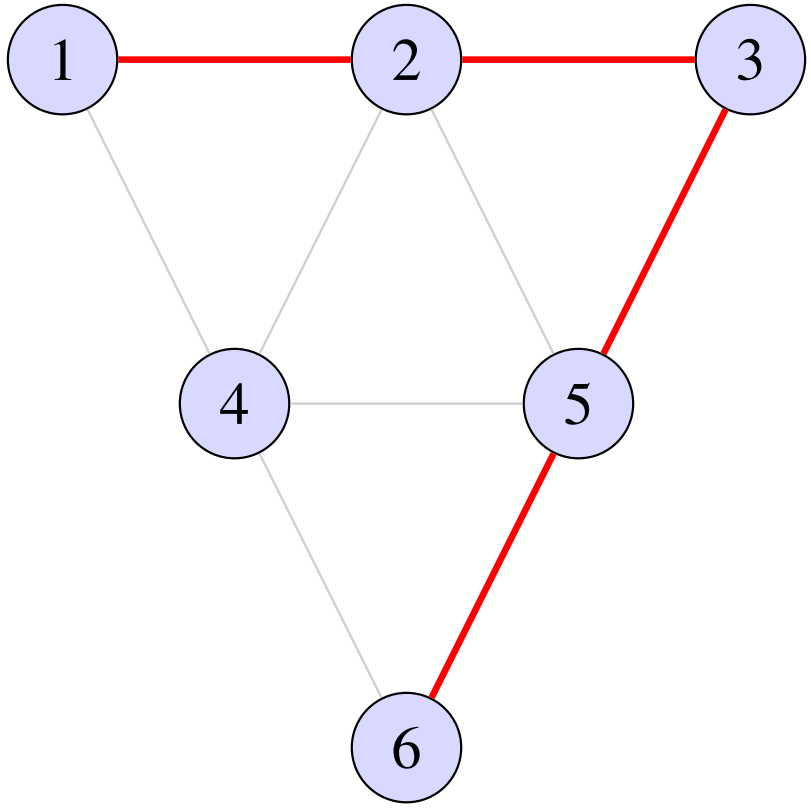
Path: $(1, 2, 3)$

Backtracking demonstration: step-by-step



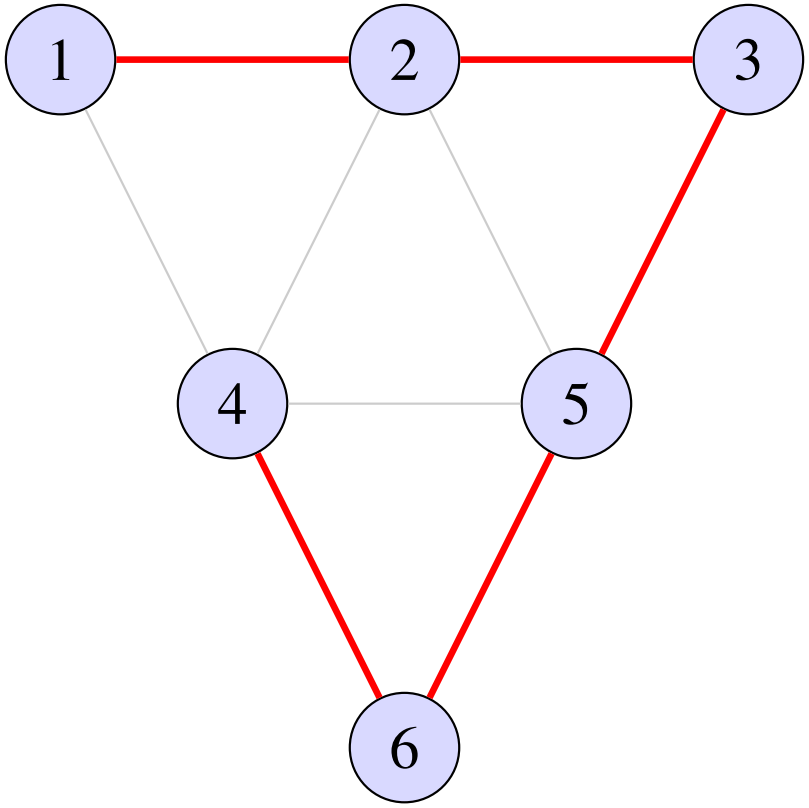
Path: $(1, 2, 3, 5)$

Backtracking demonstration: step-by-step



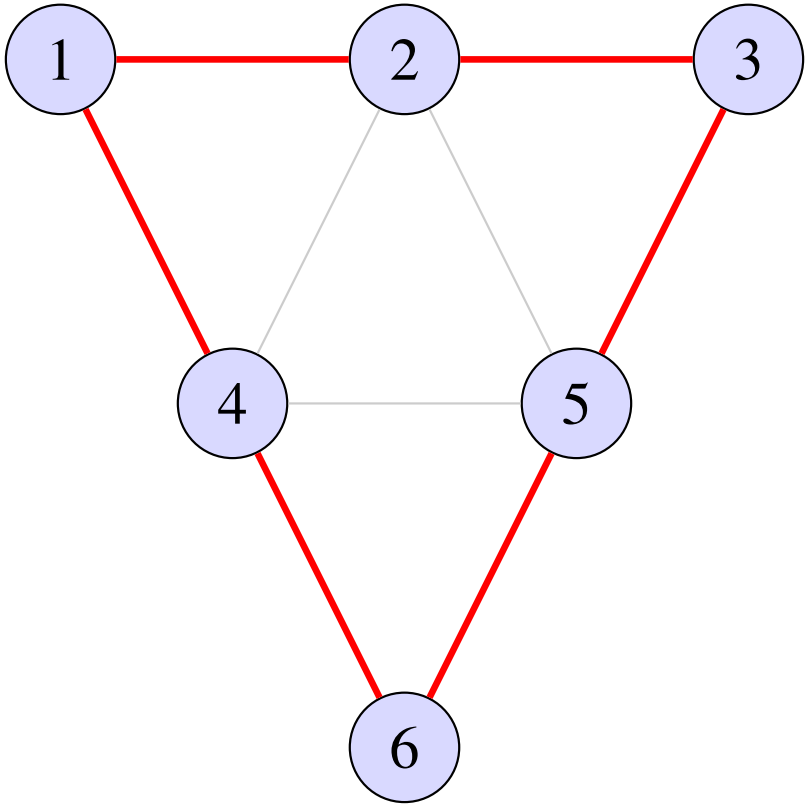
Path: $(1, 2, 3, 5, 6)$

Backtracking demonstration: step-by-step



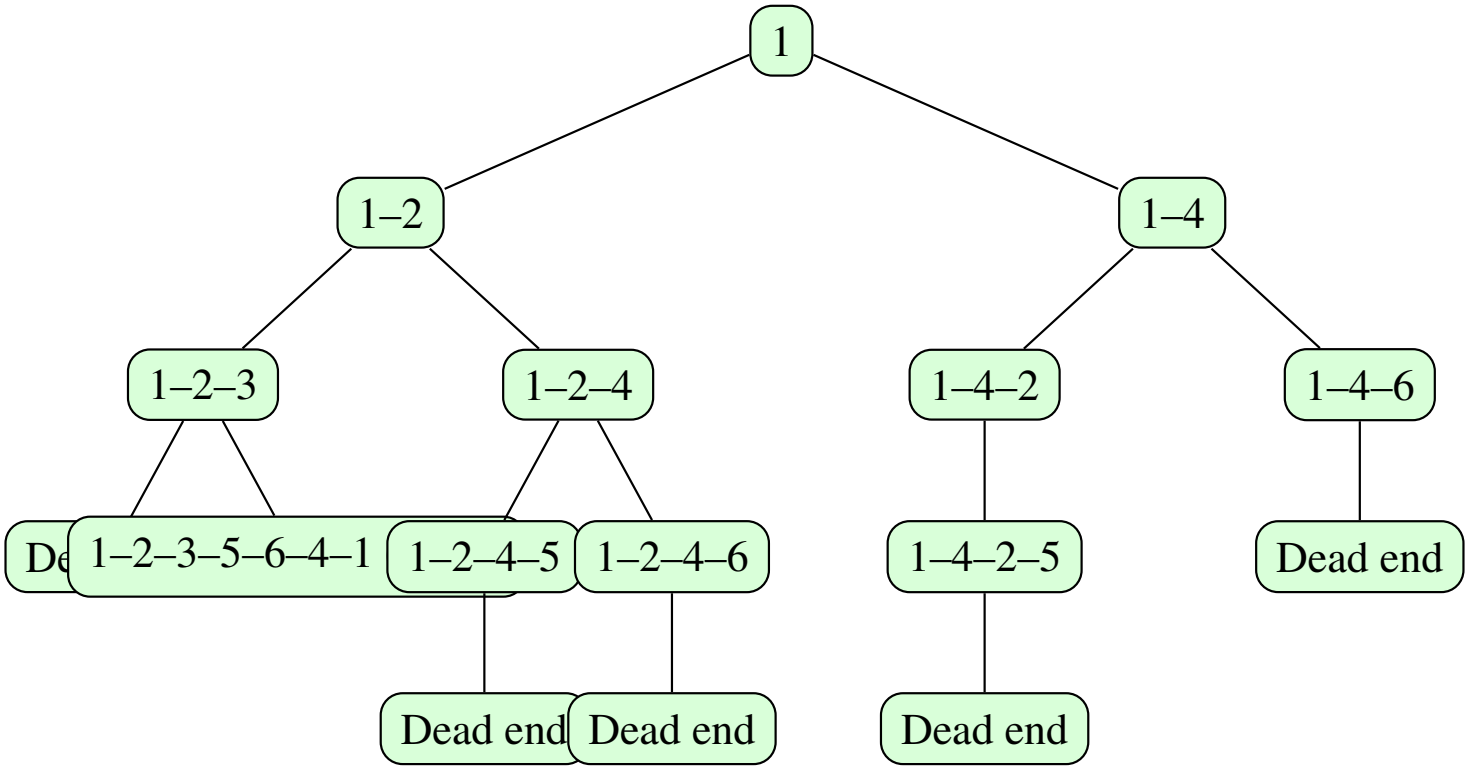
Path: $(1, 2, 3, 5, 6, 4)$

Backtracking demonstration: step-by-step



Hamiltonian cycle found: $(1, 2, 3, 5, 6, 4, 1)$

Backtracking: search tree (conceptual)



The tree shows branching choices; dead ends are pruned via backtracking until a cycle is discovered.

Held–Karp dynamic programming: idea

Held–Karp solves the *Hamiltonian cycle / TSP* exactly using DP over subsets.

State: $DP[S, v]$ = minimum cost to start at fixed start vertex (say 1), visit every vertex in subset S (which includes 1 and v), and end at vertex v.

Recurrence:

$$DP[S, v] = \min_{u \in S \setminus \{v\}} \big(DP[S \setminus \{v\}, u] + w(u, v) \big),$$

with base $DP[\{1\}, 1] = 0$.

Answer (minimum Hamiltonian cycle cost) = $\min_{v \neq 1} DP[V, v] + w(v, 1)$.

Pseudocode

1. Initialize $DP[\{1\}, 1] = 0$. For all other states set $+\infty$.
2. For subset sizes $s = 2$ to n:

   For all subsets S of size s that include 1:

   For all $v \in S$, $v \neq 1$:

   $DP[S, v] \leftarrow \min_{u \in S \setminus \{v\}} (DP[S \setminus \{v\}, u] + w(u, v))$.
3. Return $\min_{v \neq 1} DP[V, v] + w(v, 1)$.

Held–Karp: complexity and remarks

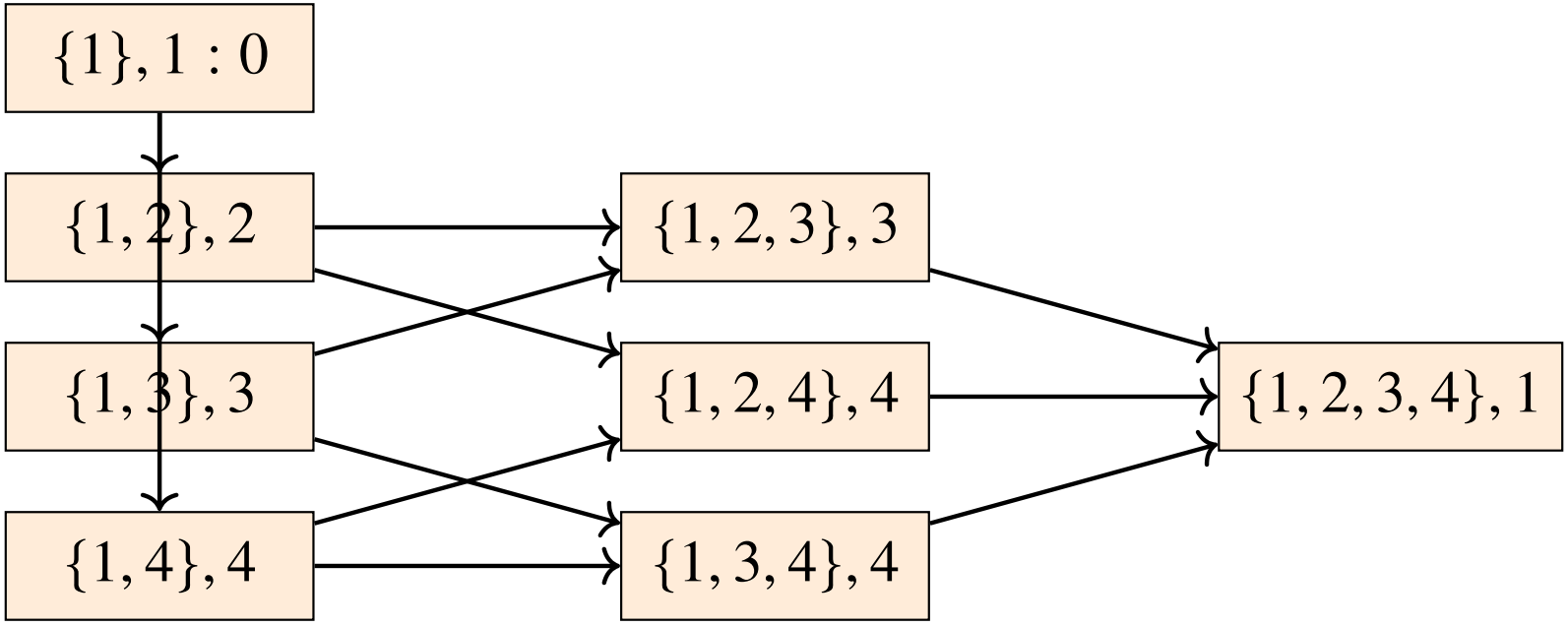
Time complexity: $O(n^2 2^n)$ — there are 2^n subsets and $O(n^2)$ work per subset across all end vertices.

Space complexity: $O(n2^n)$ to store DP table.

Practical for up to $n \approx 20$–24 (depending on memory/time), better than full $n!$ enumeration.

Works for weighted graphs (TSP); for unweighted Hamiltonian decision you can use it with unit weights or treat costs uniformly.

Illustration of DP states (small example)



Each arrow represents using the recurrence to extend a smaller subset solution into a larger one.

Complexity recap and practical comments

Decision problem: Is there a Hamiltonian cycle? — **NP-complete**.

Backtracking: worst-case $O(n!)$ but often much less with pruning.

Held–Karp: $O(n^2 2^n)$ time, $O(n 2^n)$ space.

Heuristics: for very large instances use approximate methods (Christofides, genetic algorithms, simulated annealing).

Why study Hamiltonian graphs?

They model central optimization problems (TSP), connecting graph theory to combinatorial optimization.

Serve as canonical NP-complete examples — useful for teaching complexity theory.

Practical relevance in networking, logistics, robotics, and bioinformatics.

Knowledge of both exact (DP/backtracking) and heuristic approaches is important for algorithm design.

Summary

Hamiltonian cycles visit each vertex once — different from Eulerian cycles (edges).

Existence checking is NP-complete; exact algorithms are exponential (backtracking, Held–Karp).

Held–Karp gives a practical exact approach for moderate n with complexity $O(n^2 2^n)$.

Heuristics and approximations are essential in large-scale applications.

References

J. A. Bondy and U. S. R. Murty, *Graph Theory with Applications*. Springer.

D. B. West, *Introduction to Graph Theory*.

M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*.

T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein, *Introduction to Algorithms* — for DP techniques.