Welcome to "Little Bits to Big Ideas"

Lab 5: Graphs, Intractability, and Incomputability

TAs: Ashish & Shivani

CAs: Abhinav, Harry, & Sherry

ILLINOIS

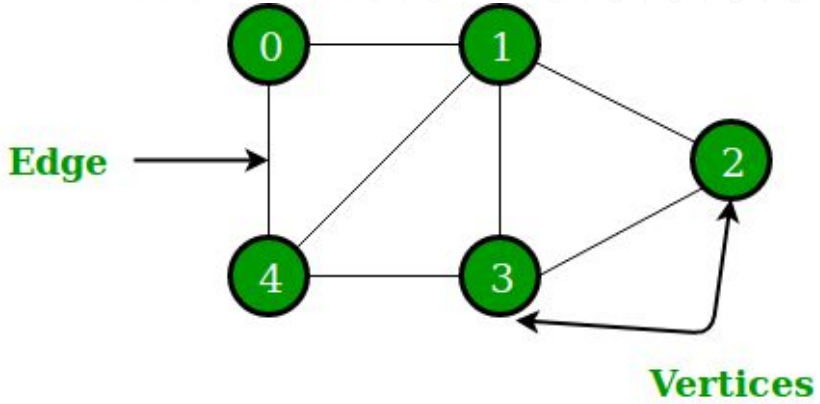
Big ideas about Intractability

- Some problems take a reasonable amount of time to solve, others take forever!
- Intractable problems are problems that cannot be solved efficiently.
- These problems grow too fast in complexity as input size increases.
- Example: Cracking a password with brute force

NP-complete problems

- **Some problems are easy to solve (P) : O(n), O(n^2), O(n log n), etc.**
  - Example: Sorting names alphabetically.
- **Some problems are hard to solve but easy to check (NP) : O(n^2), etc.**
  - Example: Solving a Sudoku puzzle. Checking a solution is quick, but finding it takes time.
- **NP-Complete = The Hardest Problems in NP : O(n^2), etc.**
  - Solutions can be checked quickly.
  - Finding a fast solution would solve all NP problems efficiently!
- **Example: Traveling Salesman Problem (TSP)**
  - Find the shortest route visiting multiple locations.
  - Checking a given route is easy, but finding the best one is extremely difficult.

Graphs in computing

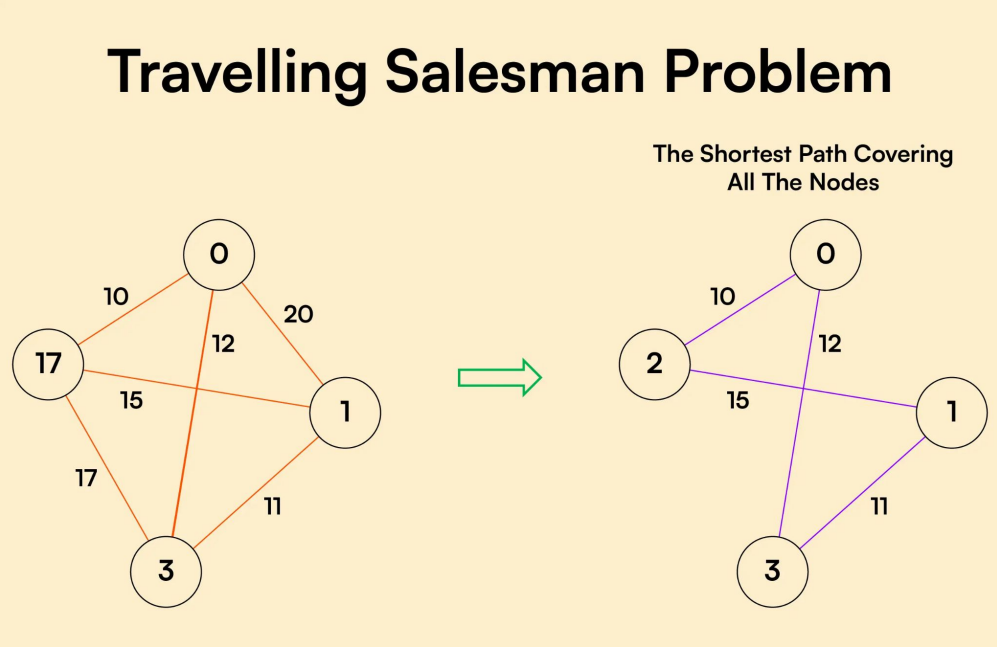


- **Graphs** = A way to represent relationships.
- **Nodes (vertices):** Represent objects.
- **Edges:** Represent connections/ relationships between objects.
- **Example:** Cities (nodes) connected by roads (edges).

Traveling Salesman Problem

If you've got a collection of places that you need to visit, and you know the distance to travel between each pair of places, what's the shortest route that visits all of the places exactly once?

- **Goal**: Find the shortest route that visits every place once and returns to the start.
- **Applications:** Logistics, circuit design, DNA sequencing.
- **Time Complexity:** Exponential $O(n^2 * 2^n)$ –intractable for large inputs.



Intro to Collatz Conjecture

The **Collatz conjecture** is one of the most famous unsolved problems in mathematics. The conjecture asks whether repeating two simple arithmetic operations will eventually transform every positive integer into 1.

- Start with any number **n**.
- If **n** is even: divide by 2. (n/2)
- If **n** is odd: multiply by 3, add 1. (3n+1)
- Repeat until reaching 1.

The Modulo operator

- **Modulo (%)** finds the remainder after division. Used to determine if a number is even or odd.

- Example: 10 % 3 = 1 (since 10 ÷ 3 leaves a remainder of 1)

- **Checking Even/Odd:**
  - If a number **n** divided by 2 has no remainder, it is even.
  - If a number **n** divided by 2 leaves a remainder of 1, it is odd.

Today you're once again using Google Co-Lab to run Python Code

This is a type of Notebook