Introduction to Graphs

CS 351: Analysis of Algorithms

Lucas P. Cordova, Ph.D.

This lecture covers the introduction to graphs, including their definition, types, and applications. We will explore the different types of graphs, their properties, and their applications.

Table of contents

0.1 Executive Summary

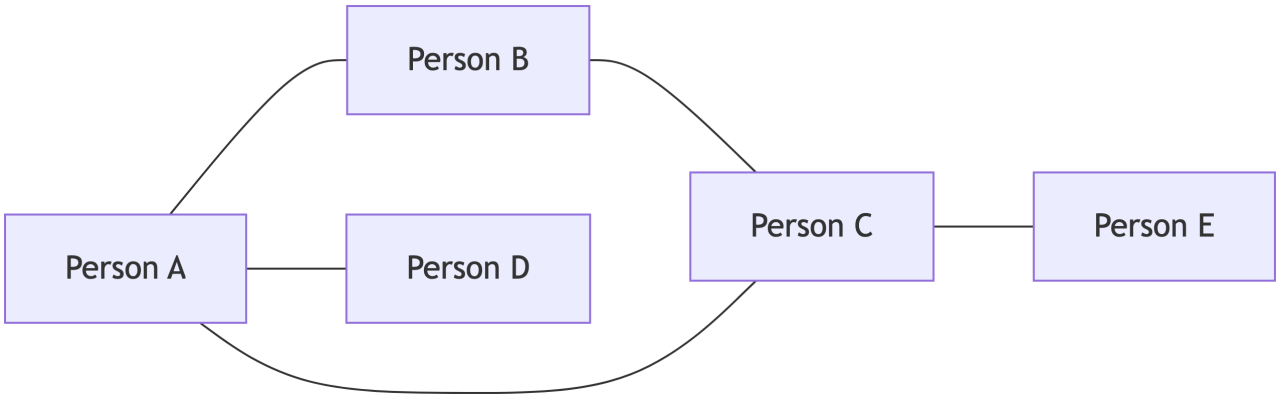
What We'll Discuss This Week

- Introduction to graphs
- Definition of graphs
- Types of graphs
- Applications of graphs

1 Introduction to Graphs

1.1 What Are Graphs?

When you hear "graph," you might think of x-y plots, but in computer science, graphs represent relationships between objects.



Key Insight: Graphs model pairwise relationships between objects

1.2 Two Types of "Graphs"

Mathematical Plot

- x-axis and y-axis
- Functions: f(n) = n, f(n) = log n
- Continuous relationships

Graph Data Structure

- Vertices (nodes) = objects

- Edges = relationships
- Discrete connections

2 Graph Fundamentals

2.1 Essential Vocabulary

Vertices (or Nodes):

- The objects being represented
- Examples: people, intersections, web pages

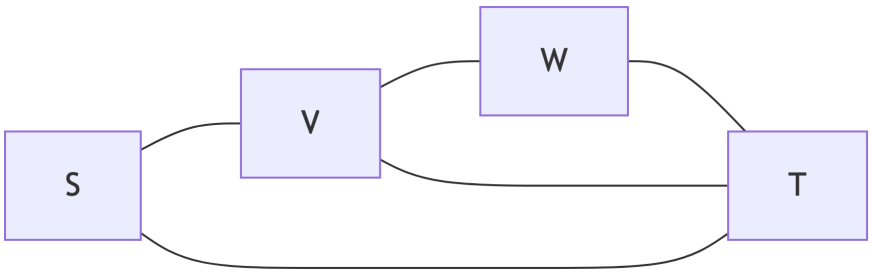
Edges:

- The pairwise relationships
- Examples: friendships, roads, hyperlinks

Notation: G = (V, E) where V = vertices, E = edges

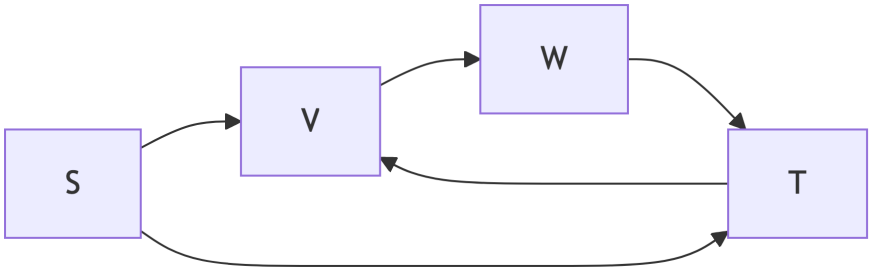
2.2 Directed vs Undirected Graphs

Undirected Graph



- Edges are unordered pairs {v, w}
- No direction matters
- (v, w) = (w, v)

Directed Graph



- Edges are ordered pairs (v, w)
- Direction matters: v → w
- v = tail, w = head

2.3 Graph Notation Deep Dive

For any graph G = (V, E):

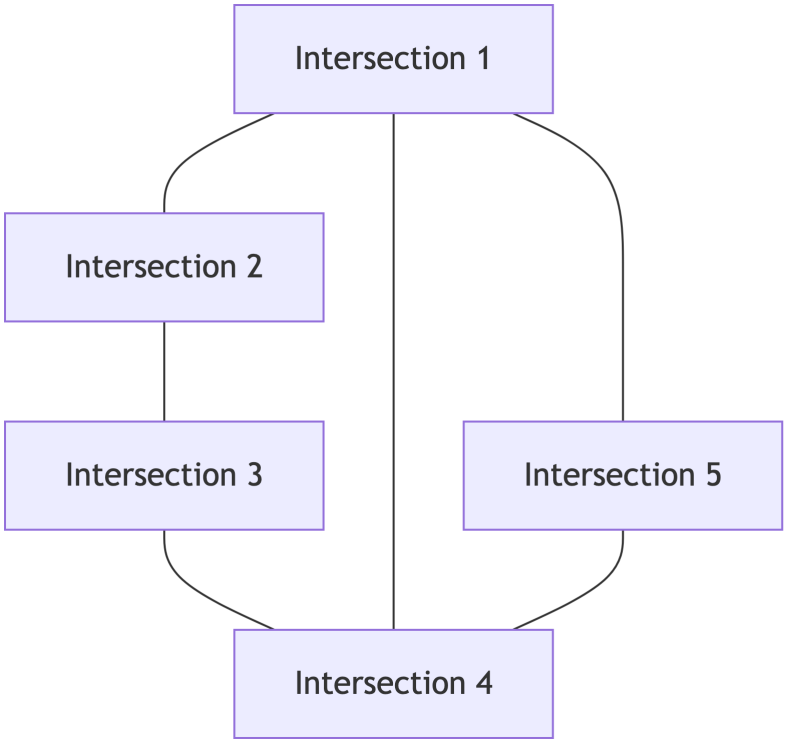
- |V| = n (number of vertices)
- |E| = m (number of edges)
- Vertices often labeled: v , v , …, v
- Edges connect vertex pairs

Example: Social network graph

- V = {Alice, Bob, Carol, Dave}
- E = {(Alice,Bob), (Bob,Carol), (Alice,Carol)}

3 Real-World Applications

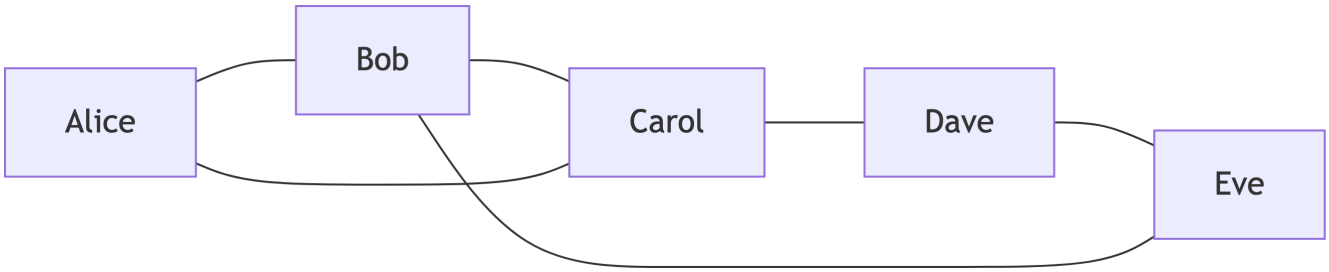
3.1 Road Networks



Implementation:

- Vertices = intersections
- Edges = road segments
- Used by GPS navigation systems

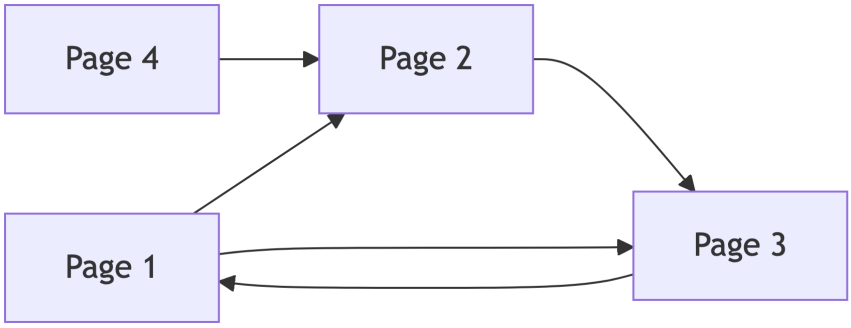
3.2 Social Networks



Applications:

- Friend recommendations
- Information spread analysis
- Community detection

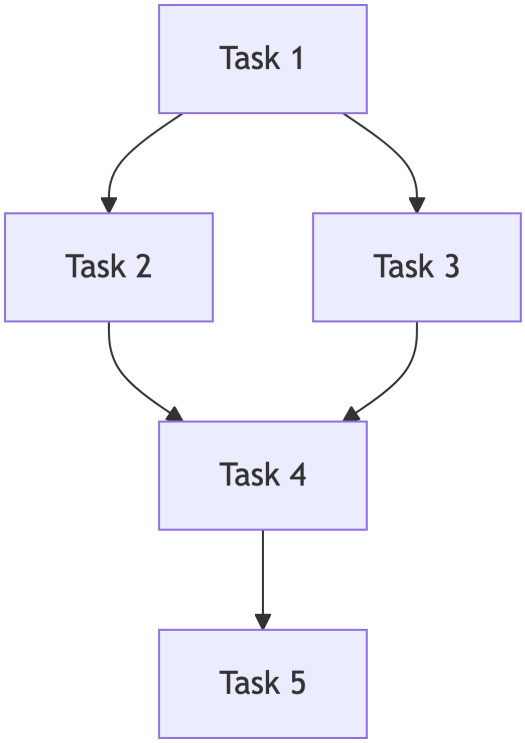
3.3 Web Structure



Key Features:

- Vertices = web pages
- Edges = hyperlinks
- Foundation of PageRank algorithm

3.4 Dependency Networks



Use Cases:

- Project scheduling
- Software compilation
- Course prerequisites

4 Graph Size Analysis

4.1 Measuring Graph Size

Two key metrics:

- n = number of vertices
- m = number of edges

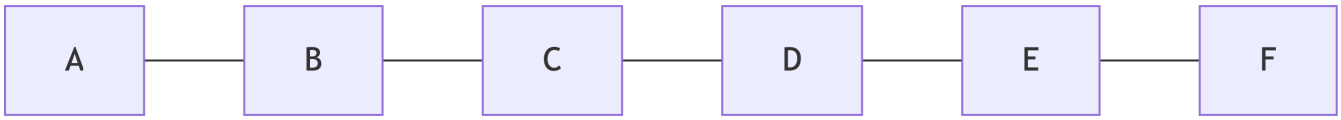
Fundamental bounds:

- Minimum edges: m 0

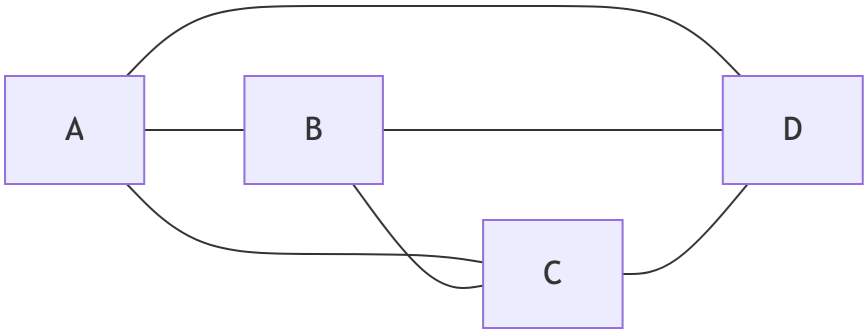
- Maximum edges (undirected): m n(n-1)/2
- Maximum edges (directed): m n(n-1)

4.2 Sparse vs Dense Graphs

Sparse Graph: m O(n)



Dense Graph: m $O(n^2)$



4.3 When Is a Graph Dense?

Mathematical Definition:

- Sparse: m = O(n) or m = O(n log n)
- Dense: m = $\Theta(n^2)$

Practical Examples:

- Social networks: typically sparse
- Complete graphs: always dense
- Road networks: usually sparse

5 Graph Representation

5.1 Two Main Approaches

Adjacency Lists:

- Space efficient for sparse graphs
- Fast edge iteration
- Preferred for most algorithms

Adjacency Matrix:

- Space efficient for dense graphs

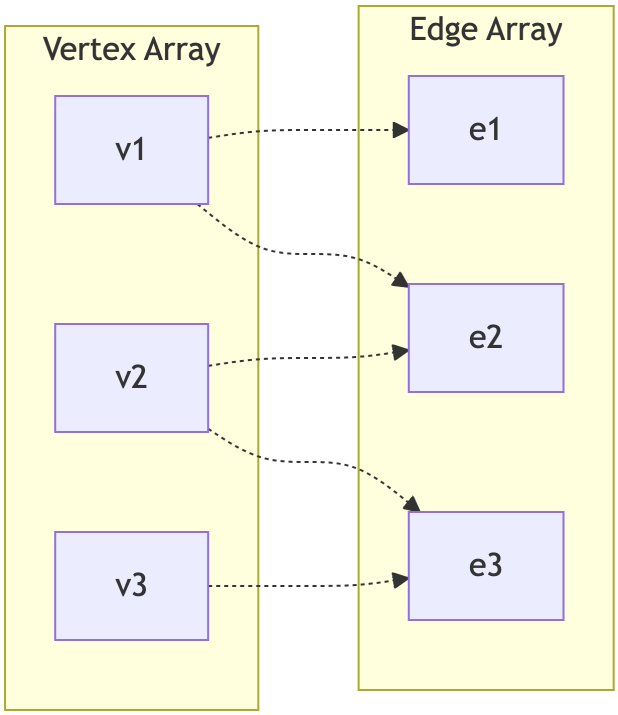
- Fast edge lookup
- Simple implementation

5.2 Adjacency List Representation

Core Components:

1. **Vertex Array**: stores vertex information
2. **Edge Array**: stores edge information

3. **Edge → Vertex pointers**: each edge points to endpoints
4. **Vertex → Edge pointers**: each vertex points to incident edges



5.3 Adjacency List: Space Analysis

Space Complexity: $\Theta(m + n)$

Breakdown:

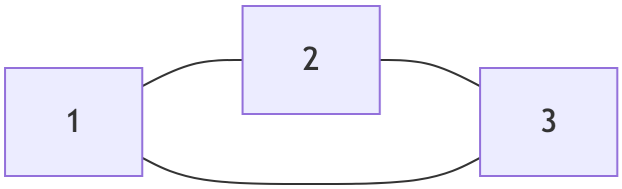
- Vertex array: $\Theta(n)$
- Edge array: $\Theta(m)$

- Edge→vertex pointers: $\Theta(m)$
- Vertex→edge pointers: $\Theta(m)$

Total: $\Theta(n + m + m + m) = \Theta(m + n)$

5.4 Adjacency List Example

Graph:



Adjacency List Representation:

- Vertex 1: [2, 3]
- Vertex 2: [1, 3]
- Vertex 3: [1, 2]

Each vertex stores list of neighbors

5.5 Adjacency Matrix Representation

Structure: n × n matrix A where:

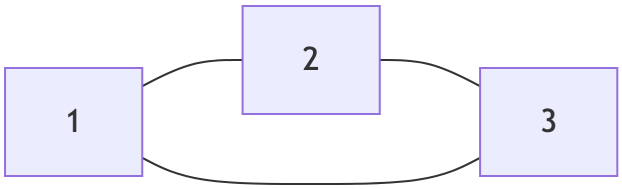
- A[i,j] = 1 if edge (i,j) exists
- A[i,j] = 0 if no edge

Undirected graphs: A[i,j] = A[j,i] (symmetric)

Directed graphs: A[i,j] A[j,i] generally

5.6 Adjacency Matrix Example

Graph:



Matrix Representation:

```
    1  2  3
1 [ 0  1  1 ]
2 [ 1  0  1 ]
3 [ 1  1  0 ]
```

5.7 Adjacency Matrix: Space Analysis

Space Complexity: $\Theta(n^2)$

Characteristics:

- Always uses n^2 bits regardless of edge count
- Efficient for dense graphs (m n^2)
- Wasteful for sparse graphs (m n)
- Fast edge existence queries: O(1)

6 Comparison and Trade-offs

6.1 Adjacency Lists vs Adjacency Matrix

Operation	Adjacency List	Adjacency Matrix
Space	$\Theta(m + n)$	$\Theta(n^2)$
Edge lookup	O(degree(v))	O(1)
Add edge	O(1)	O(1)
Remove edge	O(degree(v))	O(1)
Find neighbors	O(degree(v))	O(n)

6.2 When to Use Each Representation

Use Adjacency Lists when:

- Graph is sparse (m n)
- Need to iterate over neighbors frequently
- Memory is limited
- Most graph algorithms prefer this

Use Adjacency Matrix when:

- Graph is dense (m n^2)
- Need fast edge existence queries
- Working with mathematical graph operations
- Graph size is small and fixed

6.3 Implementation Considerations

Adjacency Lists:

- Dynamic arrays or linked lists
- Easy to add/remove vertices
- Cache-friendly for sparse graphs
- Standard choice for most algorithms

Adjacency Matrix:

- Fixed-size 2D array
- Simple bit operations
- Good for mathematical computations
- Parallel processing friendly

7 Quiz Time!

7.1 Quiz Question 1

For an undirected graph with n vertices, what is the maximum possible number of edges?

A) n

B) n - 1

C) n(n-1)/2

D) n^2

7.2 Quiz Question 2

Which representation is better for a sparse graph with 1000 vertices and 1500 edges?

A) Adjacency Matrix

B) Adjacency List

C) Both are equivalent

D) Depends on the operations

7.3 Quiz Question 3

In an adjacency list representation, how many operations are needed to check if edge (u,v) exists?

A) O(1)

B) O(log n)

C) O(degree(u))

D) O(n)

8 Practical Applications

8.1 Algorithm Performance Impact

Graph Search Algorithms:

- Breadth-First Search: O(m + n)
- Depth-First Search: O(m + n)

- Both prefer adjacency lists

Shortest Path Algorithms:

- Dijkstra's: O((m + n) log n) with heaps
- Works with both representations
- Adjacency lists usually faster

8.2 Memory Usage in Practice

Real-world example: Facebook social graph

- ~3 billion users (vertices)
- ~200 billion friendships (edges)
- Average degree: ~130
- Adjacency matrix: 9×10^1 bits 1 exabyte
- Adjacency list: ~400 billion entries 1.6 TB

Clear winner: Adjacency lists!

8.3 Graph Processing Libraries

Common implementations:

- NetworkX (Python): adjacency lists
- igraph (R/Python): adjacency lists

- GraphX (Spark): distributed adjacency lists
- SNAP (C++): adjacency lists

Industry standard: Adjacency lists with optimizations

9 Advanced Topics Preview

9.1 What's Next?

Graph Search Algorithms (Chapter 8):

- Breadth-First Search (BFS)
- Depth-First Search (DFS)
- Connected Components
- Topological Sort

Shortest Path Algorithms (Chapter 9):

- Dijkstra's Algorithm
- Single-source shortest paths
- Applications to routing

9.2 Graph Algorithm Complexity

All upcoming algorithms are "for-free primitives":

- Running time: O(m + n)
- Just slightly more than reading input
- Can apply liberally to understand graph structure

Key insight: Linear-time graph algorithms are incredibly powerful!

9.3 Modern Graph Challenges

Big Data Graphs:

- Distributed storage and computation
- Streaming graph algorithms
- Approximate algorithms for massive graphs

Machine Learning on Graphs:

- Graph Neural Networks
- Node embeddings
- Graph classification

10 Summary

10.1 Key Takeaways

Graph Fundamentals:

- Graphs model pairwise relationships
- Directed vs undirected matters
- Vertices = objects, Edges = relationships

Representation Trade-offs:

- Adjacency lists: $\Theta(m + n)$ space, prefer for sparse graphs
- Adjacency matrix: $\Theta(n^2)$ space, prefer for dense graphs

- Most algorithms use adjacency lists

10.2 Design Principles

When designing with graphs:

1. **Identify** vertices and edges clearly
2. **Choose** representation based on density
3. **Consider** the operations you'll need most
4. **Remember** adjacency lists are usually best
5. **Think** about scalability early

10.3 Next Steps

Build on this foundation:

- Practice implementing both representations
- Study graph search algorithms
- Explore real-world graph datasets
- Consider distributed graph processing
- Learn graph visualization techniques