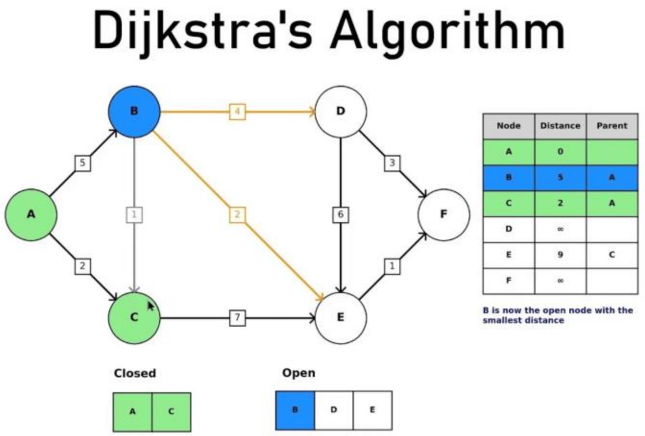
Dijkstra's Algorithm Implementation

Shortest Path Algorithm with Custom Data Structures

COP 4530 Project 4

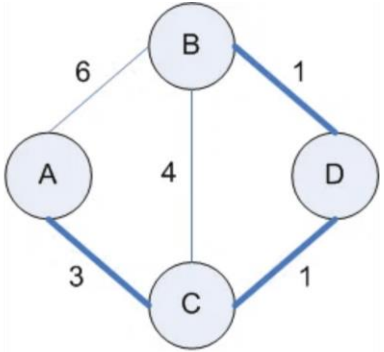


Project Goal

- Build an **Undirected Weighted Graph ADT** with vertex and edge management
- Implement **Dijkstra's Algorithm** to find shortest paths between vertices
- Return both the **total path weight** and the **vertex sequence**

⚡ Key Constraint

Priority Queue must be custom-implemented (no standard library)



ACDB, with total weight 5, is the shortest path from A to B

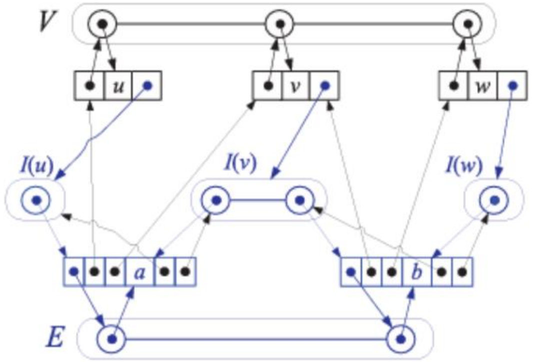
Graph Design

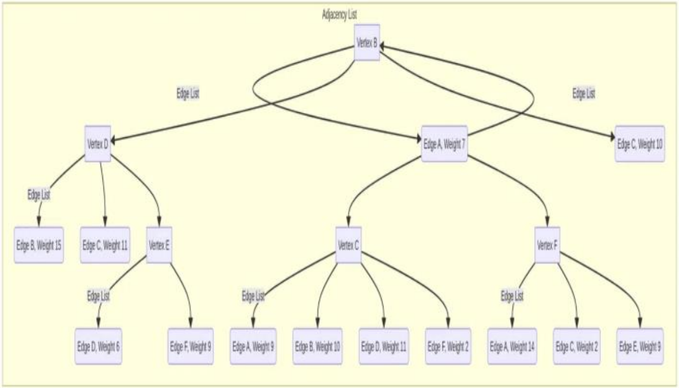
→ Use **Adjacency List** for efficient sparse graph representation

→ Create separate **Vertex** and **Edge** classes for OOP design

→ Implement proper **destructors** for memory management

> 💡 **Best Practice**
>
> Separate headers (.hpp) and source (.cpp) files for clean architecture





Core Methods

addVertex()

Create and add a vertex with a unique label

removeVertex()

Remove a vertex and all its connected edges

addEdge()

Add a weighted edge between two vertices

removeEdge()

Remove an edge between two vertices

shortestPath()

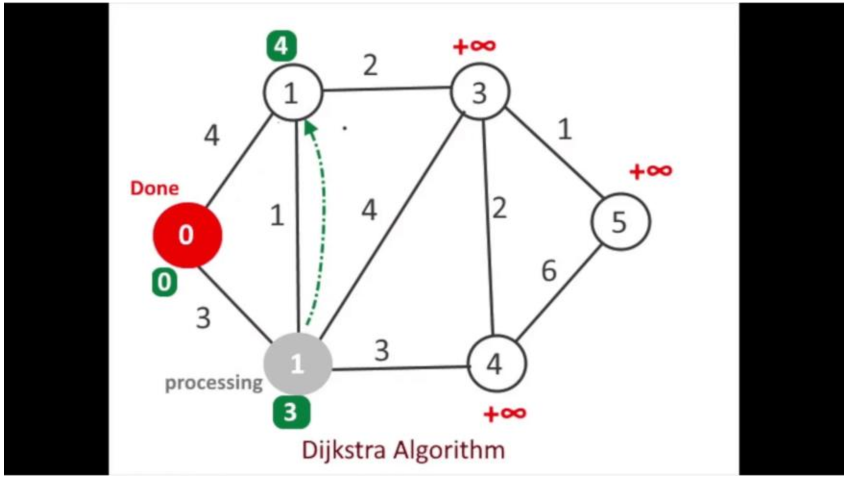
Find shortest path using Dijkstra's Algorithm

Time Complexity & Space Complexity

Operation	***Time***
vertices	$O(n)$
edges	$O(m)$
endVertices, opposite	$O(1)$
v.incidentEdges()	$O(\deg(v))$
v.isAdjacentTo(w)	$O(\min(\deg(v), \deg(w))$
isIncidentOn	$O(1)$
insertVertex, insertEdge, eraseEdge,	$O(1)$
eraseVertex(v)	$O(\deg(v))$

Dijkstra's Algorithm

1. **Initialize:** Set all distances to infinity, start node to 0
2. **Select:** Extract the unvisited vertex with minimum distance
3. **Relax:** Update distances to neighboring vertices if shorter path found
4. **Reconstruct:** Build the path using predecessor information

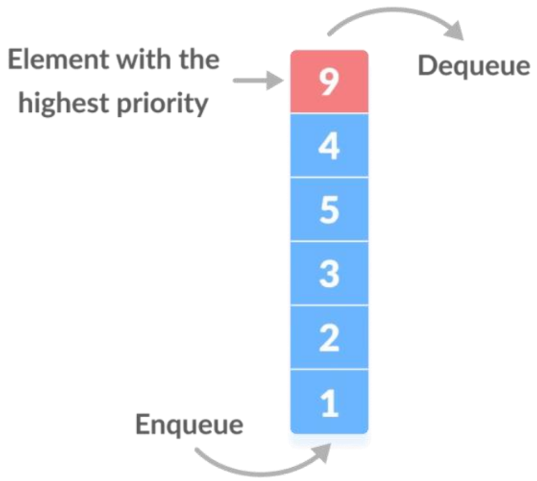


Priority Queue

🔴 Critical Requirement

Must be custom-implemented. No standard library PQ allowed.

- → **push()** — Add element with priority
- → **pop()** — Get lowest-cost vertex
- → **bubbleUp/Down** — Maintain heap property



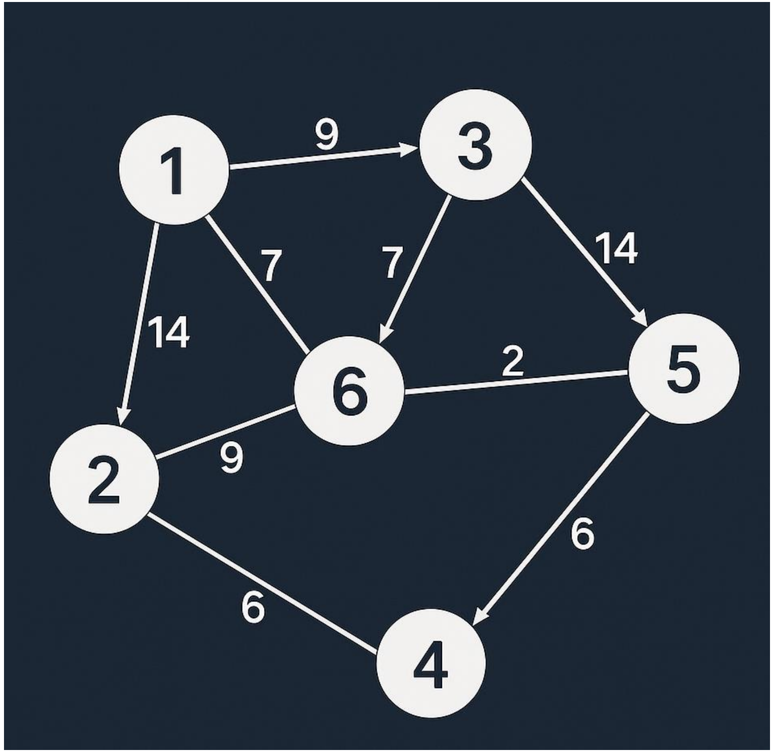
Testing

→ **Correctness (Expected outputs match known solutions):**
  - → Provided example graph → *Cost = 20, Path = 1 → 3 → 6 → 5*
  - → Directly connected vertices → *Correct cost & path*

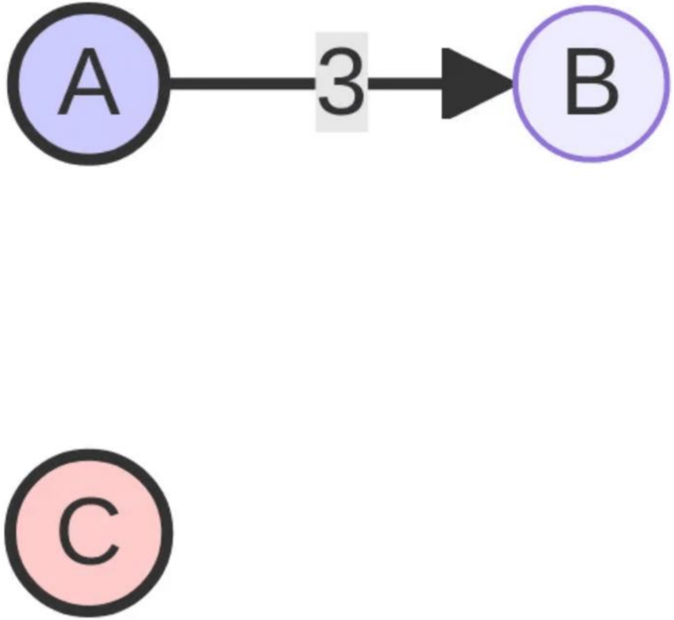
→ **Edge Cases:**
  - → No path between vertices → correctly returns **infinity**
  - → Multiple shortest paths → algorithm returns one valid optimal path
  - → Single vertex / isolated nodes

→ **Graph Modification Behavior:**
  - → **Removing edges** changes shortest path as expected
  - → **Removing vertices** breaks connectivity correctly
  - → Graph still runs Dijkstra properly after structural changes

→ **Performance & Scaling:**
  - → Chain of **10 vertices** tested → correct path of length 9
  - → Ensures O((V + E) log V) behavior holds for larger inputs



Test Case 1: No Path Exists



GOAL

Find the shortest path from vertex **A** to vertex **C**

SETUP

Graph contains vertices A, B, and C. Edge A→B has weight 3. Vertex C is **isolated** (no edges connected to it).

EXPECTED RESULT

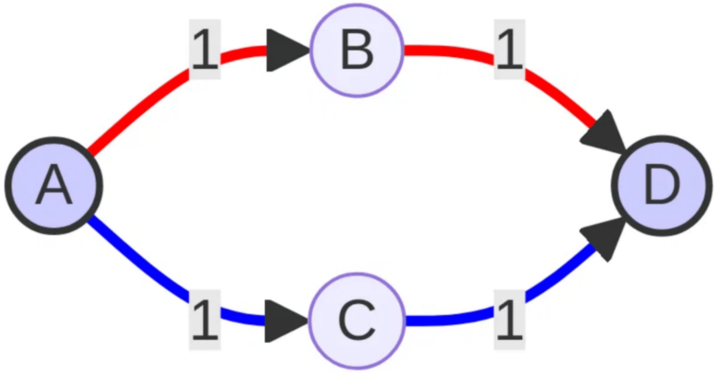
Cost = ∞

Path = empty

✓ VALIDATION

This test confirms that the algorithm correctly handles **disconnected components** and properly terminates when no path exists between the source and destination vertices.

Test Case 2: Multiple Equal Shortest Paths



GOAL

Test tie-breaking mechanism when multiple paths share the same minimum cos

SETUP

Graph with two equal-cost paths from A to D

PATHS

1 A → B → D (Cost: 2)

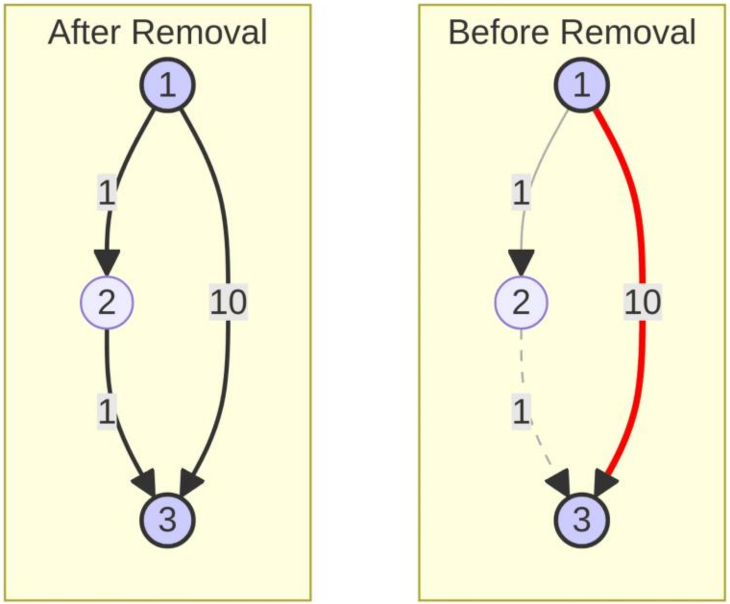
2 A → C → D (Cost: 2)

EXPECTED RESULT

Cost = 2. Algorithm returns one of the two optimal paths

> **Validation:** Confirms the algorithm finds an optimal solution even with path ambiguity. Tie-breaking behavior is consistent and deterministic.

Test Case 3: Edge Removal



Goal

Test dynamic graph modification and its impact on shortest path computation

Setup

Graph with three vertices (1, 2, 3) and three edges:

> 1 → 2 (weight: 1)
> 2 → 3 (weight: 1)
> 1 → 3 (weight: 10)

Before Removal

Shortest path: **1 → 2 → 3**
Cost: **2** (1 + 1)

Action

Remove edge: **2 → 3**

After Removal

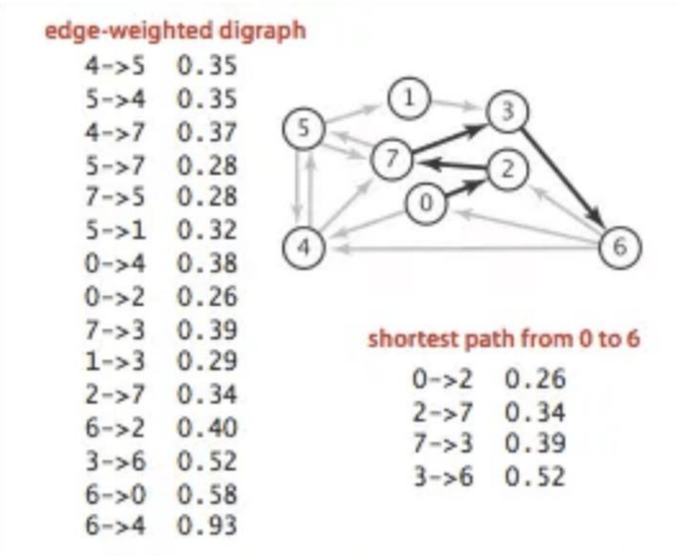
Shortest path: **1 → 3**
Cost: **10**

Validation

✓ Confirms correct **removeEdge()** implementation
✓ Verifies Dijkstra's re-computation on modified graph
✓ Demonstrates graph structural integrity

Code Implementation

```
Graph g;
 // Add vertices
g.addVertex("1");
g.addVertex("2");
g.addVertex("3");
g.addVertex("4");
g.addVertex("5");
g.addVertex("6");
// Add weighted edges
g.addEdge("1", "2", 7);
g.addEdge("1", "3", 9);
g.addEdge("1", "6", 14);
g.addEdge("2", "3", 10);
g.addEdge("2", "4", 15);
g.addEdge("3", "4", 11);
g.addEdge("3", "6", 2);
g.addEdge("4", "5", 6);
g.addEdge("5", "6", 9);
```



An edge-weighted digraph and a shortest path

Live Execution

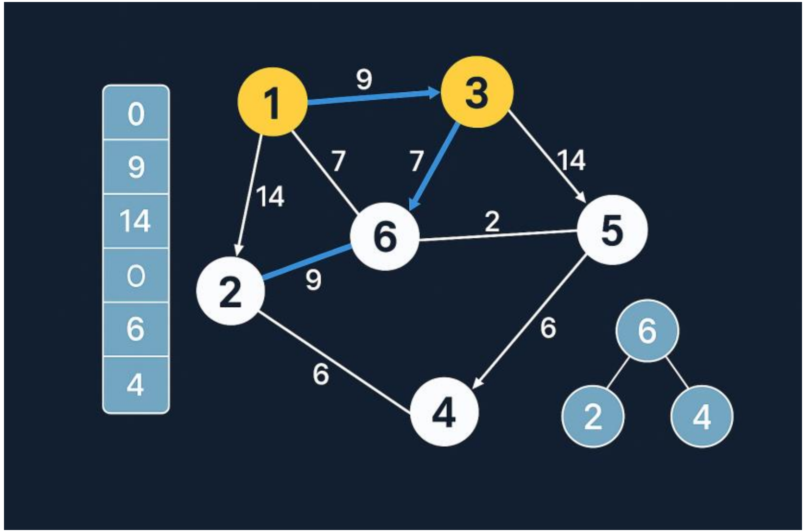
```
std::vector<std::string> path;

unsigned long cost = g.shortestPath("1", "5", path);
```

Shortest Path Cost: 20

Path: 1 → 3 → 6 → 5

Algorithm executed successfully!



Team Member 1

Saksham Srivastava

Lead Developer

- Designed and implemented the **Graph ADT** architecture and implemented **Dijkstra's Algorithm** with full optimization
- Developed the **Vertex and Edge** classes with full functionality
- Implemented core graph operations: **addVertex, removeVertex, addEdge, removeEdge**

Team Member 2

Diego Rey Martinez

Test Case Handler

→ Developed **path reconstruction** logic for result generation

→ Optimized algorithm performance and edge case handling

→ References and research for **design**

Team Member 3

Phuc Truong

Data Structures Engineer

- Wrote test cases for our programs
- Developed **bubbleUp** and **bubbleDown** heap operations
- Debug and fix bug for our codebase
- Wrote report and slides for presentation

Team Member 4

Diego Laverdy

QA & Documentation Lead

→ Created comprehensive **test cases** and validation suite

→ Wrote detailed **code documentation** and comments

→ Prepared project **report .**

→ Contributed to **algorithm discussion**

→ Contributed to **presentation**

Summary

- ✓ Successfully implemented a **Graph ADT** with full vertex and edge management
- ✓ Mastered **Dijkstra's Algorithm** for efficient shortest path computation
- ✓ Built a **custom Priority Queue** from scratch for optimal performance
- ✓ Applied **object-oriented design** principles throughout the project

Questions?