# Project 2: Dijkstra's Algorithm

Group 4: Jun Meng, Kai Xin, Spencer

## Project 2: Dijkstra's Algorithm

### (a) Adjacency Matrix + Array-based Priority Queue

**Idea:** Graph stored as a $V \times V$ matrix. Priority queue implemented as a simple array; extract-min requires scanning all vertices.

**Time Complexity:**

- Extract-min: $O(V)$ per vertex $\Rightarrow O(V^2)$ total
- Relaxation: For each vertex, scan all possible neighbors $\Rightarrow O(V^2)$ total
- Total: $O(V^2)$

**Effect of $|V|$ vs $|E|$:**

- Increasing $|V|$: adds rows/columns $\Rightarrow$ runtime grows roughly quadratically
- Increasing $|E|$ (fixed $|V|$): little to no effect; all entries scanned regardless

**Space Complexity:** $O(V^2)$ - always store all possible edges.

**Empirical Observation:** Fast for small or dense graphs; runtime largely insensitive to $|E|$.

## (b) Adjacency List + Min-Heap Priority Queue

**Idea:** Graph stored as an array of adjacency lists. Priority queue implemented as a min-heap for efficient extract-min and decrease-key.

**Time Complexity:**

- Extract-min: $O(\log V)$ per vertex $\Rightarrow O(V \log V)$ total
- Relaxation (decrease-key): $O(\log V)$ per edge $\Rightarrow O(E \log V)$ total
- Total: $O((V + E) \log V) = O(V \log V + E \log V)$

**Effect of $|V|$ vs $|E|$:**

- Increasing $|V|$: more heap operations $\Rightarrow O(V \log V)$ contribution
- Increasing $|E|$: more edges to relax $\Rightarrow O(E \log V)$ contribution

**Space Complexity:** $O(V + E)$ - stores only existing edges.

**Empirical Observation:** Efficient for sparse graphs; scales well with both $|V|$ and $|E|$.

## (c) Comparison & Intuition

| **Implementation** | **Best For** | **Time Complexity** | **Space Complexity** |
|---|---|---|---|
| Matrix + Array | Dense graphs, small $V$ | $O(V^2)$ | $O(V^2)$ |
| List + Min-Heap | Sparse graphs, large $V$ | $O((V+E)\log V)$ | $O(V+E)$ |

**Intuition:**

- **Sparse graphs** ($E \ll V^2$): List + heap wins - avoids scanning non-existent edges and uses less memory.
- **Dense graphs** ($E \approx V^2$):
  - Both representations store similar amounts of data.
  - Array-based PQ is simpler and often faster in practice due to lower constant factors, despite $O(V^2)$ complexity.
  - Heap-based PQ is better for very large $V$ due to better asymptotic scaling.