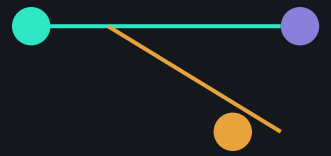


LECTURE 27

Shortest Path: Dijkstra's Algorithm

CSC211 · Algorithms and Data Structures

COMSATS University Islamabad, Lahore Campus

LECTURE 27

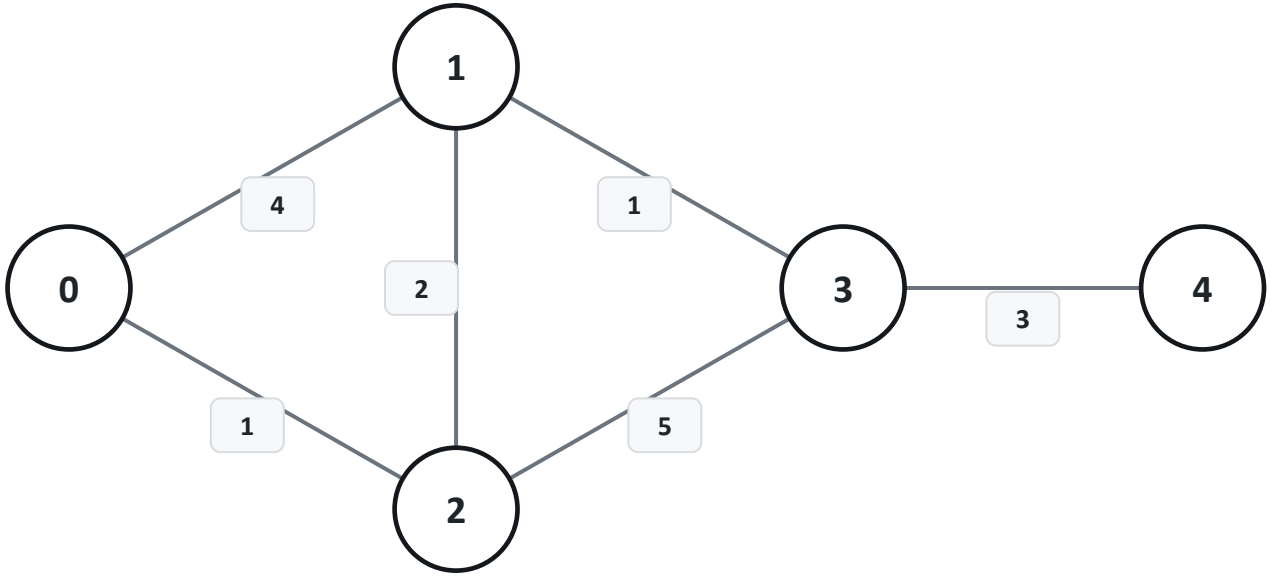
Agenda

1. The shortest-path concept and single-source shortest path
2. Dijkstra's algorithm, built on a min-heap priority queue
3. A full step-by-step trace: how distance[] evolves as vertices finalize
4. What happens on a disconnected graph
5. Time complexity, applications, and where the algorithm breaks down

SINGLE-SOURCE SHORTEST PATH

The Shortest-Path Concept

In a weighted graph, “shortest” means smallest **total weight**, not fewest edges. Find the shortest distance from one source to every other reachable vertex, all at once.




```
0 -> 1 direct: weight 4
```

0 -> 2 -> 1: 1 + 2 = 3

The shortest path from 0 to 1 is NOT the direct edge — it's the 2-hop path via vertex 2, beating the direct edge by 1. Dijkstra's algorithm finds exactly this kind of answer, systematically.

GREEDY + PRIORITY QUEUE

Dijkstra's Algorithm

Keep a running best-known distance[] to every vertex. Repeatedly pop the closest unprocessed vertex from a min-heap and **relax** its neighbors.

dijkstra.cpp — dijkstra()

```
vector<int> dijkstra(int source) const {
  vector<int> distance(numVertices, INT_MAX);
  distance[source] = 0;
  priority_queue<pair<int,int>, vector<pair<int,int>>,
                 greater<pair<int,int>>> pq;
  pq.push({0, source});

  while (!pq.empty()) {
    auto [currentDist, current] = pq.top(); pq.pop();
    if (currentDist > distance[current]) continue;
  stale entry
    for (auto& [nb, w] : adjList[current]) {
      int newDist = distance[current] + w;
      if (newDist < distance[nb]) {
        distance[nb] = newDist;
  relax
        pq.push({newDist, nb});
      }
    }
  }
  return distance;
}
```

Result: distances from vertex 0

v	0	1	2	3	4
dist	0	3	1	4	7

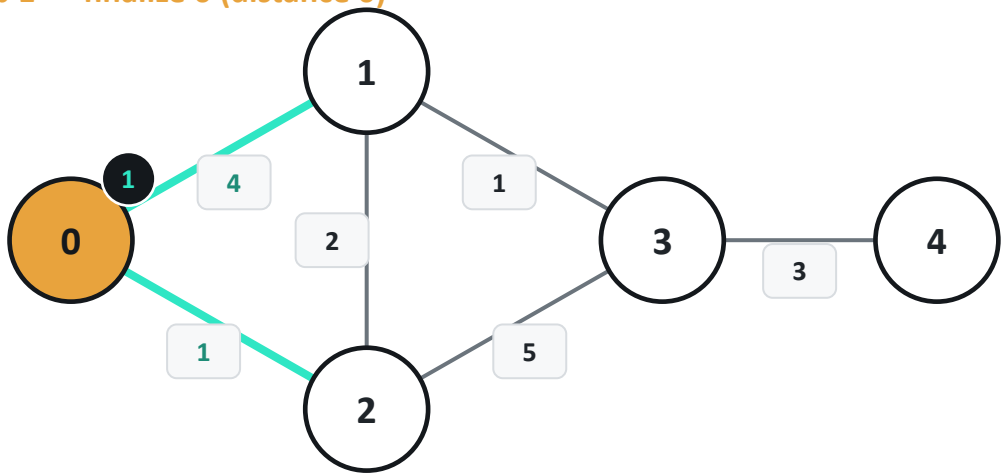
Confirmed by hand: 0→2 costs 1; 0→2→1 costs 3; 0→2→1→3 costs 4; 0→2→1→3→4 costs 7 — every distance is a real, traceable path.

Why a stale entry can exist: a shorter path can be found AFTER an older, longer entry is already queued — the continue guard skips it.

STEP-BY-STEP TRACE (1/2)

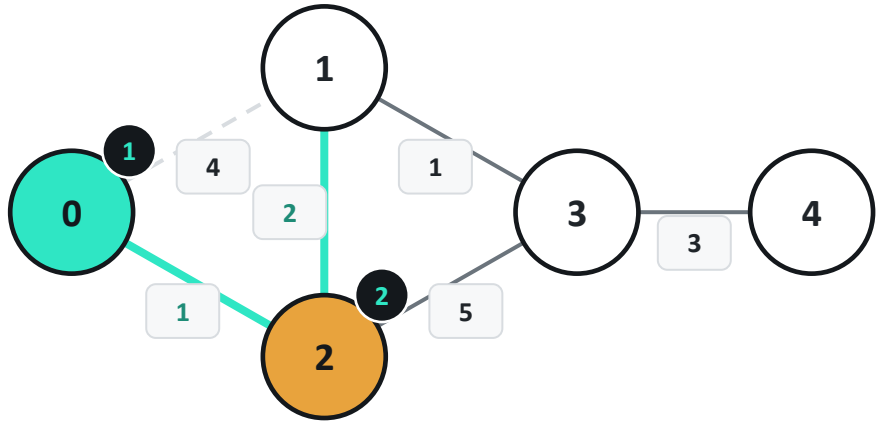
Tracing Dijkstra: Finalizing Vertex 0, Then 2

Step 1 — finalize 0 (distance 0)



v	0	1	2	3	4
dist	0	4	1	inf	inf

Step 2 — finalize 2 (distance 1)




```
relax 1: 4 -> 3 (via 0-2-1)    relax 3: inf -> 6
```

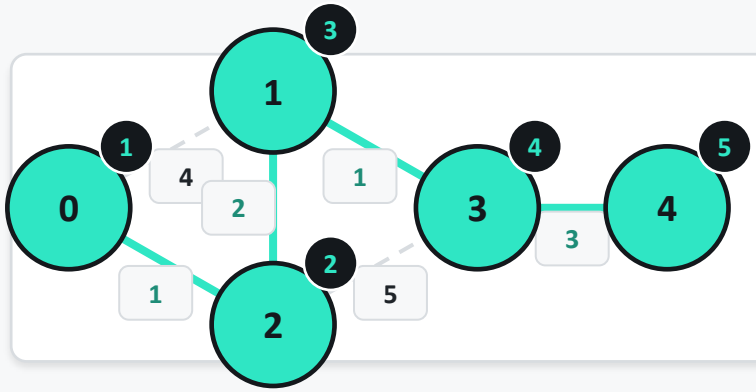
v	0	1	2	3	4
dist	0	3	1	6	inf

STEP-BY-STEP TRACE (2/2)

Tracing Dijkstra: The Full Run

Step	Vertex finalized	Its distance	distance[0..4] after
1	0	0	`[0, 4, 1, inf, inf]`
2	2	1	`[0, 3, 1, 6, inf]`
3	1	3	`[0, 3, 1, 4, inf]`
4	3	4	`[0, 3, 1, 4, 7]`
5	4	7	`[0, 3, 1, 4, 7]`

Finalize order is 0, 2, 1, 3, 4 — NOT vertex-number order. Vertex 1's entry changes twice (inf→4, then 4→3) before it's finalized; once finalized, a value never changes again.

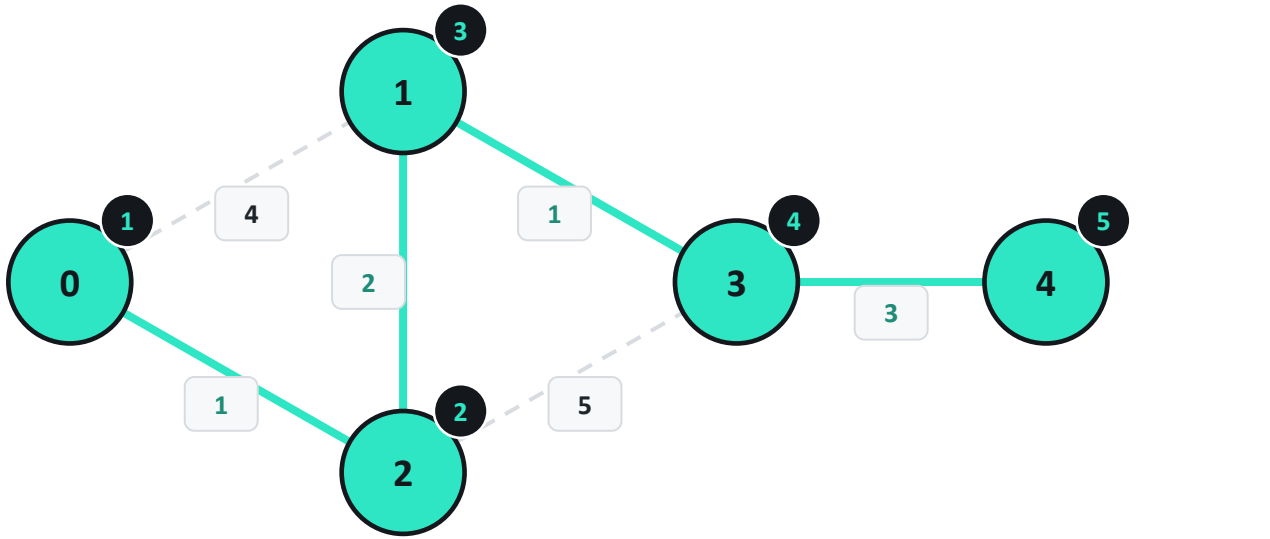


Shortest-path tree from source 0, highlighted in teal

UNREACHABLE VERTICES

What About a Disconnected Graph?

distance[] starts at **infinity** for everyone. If a vertex has no path from the source, it's simply never pushed onto the queue — no crash, no special case.



Separate component — no edge reaches {0..4}

Shortest distances from 0

```
0: 0
1: 3
2: 1
3: 4
4: 7
5: unreachable
6: unreachable
```

INT_MAX is a stand-in for infinity. Vertices 5, 6 are never dequeued — not even once — because no relaxation step ever triggers for them.

WRAPPING UP DIJKSTRA

Complexity, Applications, and Limits

O((V+E) log V)
with a binary-heap priority queue

G **GPS & mapping software**
Literally this algorithm, finding shortest routes on a weighted road network.

N **Network routing protocols**
Lowest-cost path for data packets across routers.

P **Game AI pathfinding**
Shortest route on a weighted grid or map graph.

Limitations of Dijkstra's Algorithm

Cannot handle negative edge weights.
Once a vertex is finalized, its distance can never improve — a later negative edge could violate that.

Single-source only.
Finds shortest paths from ONE start vertex — all-pairs needs a different algorithm.

KEY TAKEAWAYS

- Dijkstra's algorithm solves single-source shortest path on a weighted graph, greedily processing the closest unprocessed vertex via a min-heap priority queue.
- Relaxation — checking whether going through the current vertex improves a neighbor's known distance — is the core operation.
- Vertices finalize in increasing order of shortest distance, not vertex-number order; a distance can be relaxed multiple times before finalizing, but never after.
- On a disconnected graph, an unreachable vertex is simply never pushed onto the queue — no special-case code needed.
- Runs in O((V + E) log V), but fails on negative edge weights and only answers shortest-path-from-one-source.