PROBLEM: SINGLE SOURCE SHORTEST PATHS (SSSP)
- Input: graph \( G = (V, E) \) and a non-negative weight function \( w(e) \) defined for every edge \( e \).
- Problem: for every node \( v \neq s \), output a path \( s \rightarrow v \) with the smallest total weight (among all paths \( s \rightarrow v \))
  \[ \text{i.e., each path } P \text{ should minimize } \sum_{e \in P} w(e) \]

**Illustrative Example**

Game AI: path finding with waypoints
- Divide game world into linear paths, then send game characters straight lines between waypoints
- Some linear paths are much faster (fewer cost weighted SSSP)
- Otherwise use BFS to find shortest sequence of waypoints with fewest waypoints

DIJKSTRA’S ALGORITHM

APPLICATION: DRIVING DISTANCE TO MANY POSSIBLE DESTINATIONS
- Single source: from where you are
- Shortest path to all destinations
- Display a subset of destinations
- Include the optimal distances computed using SSSP algorithm
- Other heuristics... traffic? Lights?
- Weights can combine many factors

DIJKSTRA’S ALGORITHM

Single-source shortest path
in a graph with non-negative edge weights
**POOFS**

**Theorem:** At the end of the algorithm, for all \( u \), \( dist[u] \) is exactly the total weight of the shortest \( s \rightarrow u \) path.

- We prove this in two parts:
  - \( dist[u] \leq \) the total weight of the shortest \( s \rightarrow u \) path.
  - \( dist[u] \geq \) the total weight of the shortest \( s \rightarrow u \) path.

**Correctness:**

- Dijkstra's algorithm iteratively constructs a set \( OPT \) of nodes for which we know the shortest path from \( s \) [initially \( OPT = \{s\} \)]

After each relaxation step, we grow \( OPT \) by adding the node in \( V \setminus OPT \) with the smallest \( dist \).

**Base Case:**

- Let \( P \) be any arbitrary \( s \rightarrow u \) path \( v_0 \rightarrow v_1 \rightarrow \cdots \rightarrow v_j \)
  - where \( v_0 = s \) and \( v_j = u \)
  - For any index \( j \), let \( L_j \) denote \( d(w(v_0 \rightarrow v_1 \rightarrow \cdots \rightarrow v_j)) \)
  - We prove by induction: \( dist[v_j] \leq L_j \) for all \( j \)

**Inductive Hypothesis:**

- Suppose \( v_j \) and \( v_{j+1} \) are adjacent vertices.

**Inductive Step:**

- Define \( L_{j+1} \) as the length of the shortest \( s \rightarrow v_{j+1} \) path.
  - By I.H., \( dist[v_j] \leq L_j \) \( \left( \text{This proves \( dist[v_j] \leq L_j \)} \right) \)
  - \( dist[v_{j+1}] = w(v_j, v_{j+1}) + L_j \)

**Prove by Induction:**

- Suppose \( v_j \) and \( v_{j+1} \) are adjacent vertices.

**Correctness:**

- Dijkstra's algorithm iteratively constructs a set \( OPT \) of nodes for which we know the shortest path from \( s \) [initially \( OPT = \{s\} \)]

After each relaxation step, we grow \( OPT \) by adding the node in \( V \setminus OPT \) with the smallest \( dist \).
dist[u] ≥ SHORTEST s ⇨ u PATH

P' = v₀ → ... → vₗ = s → ... → pred[pred[u]] → pred[u] → u
Lₗ = w(v₀ → v₁ → ... → vₗ)

Inductive step: suppose ∀₀ ≤ j ≤ Lₗ : dist[vⱼ] = Lⱼ
When we set pred[vⱼ] = vⱼ₋₁, we set dist[vⱼ] = dist[vⱼ₋₁] + w(vⱼ₋₁, vⱼ)

Recall:

if dist[u] = w(u, v) ≥ dist[v] ≥ dist[u] + w(u, v)
pred[v] = u

By I.H., dist[vⱼ] = Lⱼ₋₁ + w(vⱼ₋₁, vⱼ)
By definition Lⱼ = Lⱼ₋₁ + w(vⱼ₋₁, vⱼ)
So dist[vⱼ] = length of shortest path s → u
So dist[u] ≥ length of shortest path s → u
That means it's equal to the length of the shortest path!

OUTPUTTING ACTUAL SHORTEST PATH(S)?

To compute the actual shortest path s ⇨ t

- Inspect pred[t]
  - If it is NULL, there is no such path
  - Otherwise, follow pred pointers back to s, and return the reverse of that path

WEBSITE DEMONSTRATING DIJKSTRA'S ALG


BELLMAN-FORD

Single-source shortest path in a graph with possibly negative edge weights but no negative cycles
Shortest Paths and Negative Weight Cycles

Subsequent algorithms we will be studying will solve shortest path problems as long as there are no cycles having negative weight. If there is a negative weight cycle, then there is no shortest path (why?).

There is still a shortest simple path, but there are apparently no known efficient algorithms to find the shortest simple paths in in graphs containing negative weight cycles.

If there are no negative weight cycles, we can assume WLOG that shortest paths are simple paths (any path can be replaced by a simple path having the same weight).

Negative weight edges in an undirected graph are not allowed, as they would give rise to a negative weight cycle (consisting of two edges) in the associated directed graph.

**BEST CASE EXECUTION**

Edges happen to be processed left to right by the inner loop

**WORST CASE EXECUTION**

Edges happen to be processed right to left by the inner loop

**WHY BELLMAN-FORD WORKS**

Not going to prove this (by induction), but the crucial lemma is:

After \(i\) iterations of the outer for-loop,

- if \(D[u] \neq \infty\), it is equal to the weight of some path \(s \rightarrow u\); and
- if there is a path \(P = (s \rightarrow u)\) with at most \(i\) edges, then \(D[u] \leq w(P)\)

So, after \(n - 1\) iterations, if there is a path \(P\) with at most \(n - 1\) edges, then \(D[u] \leq w(P)\). (Note: any more edges would create a cycle.)

So, if \(u\) is reachable from \(s\), then \(D[u]\) is the length of the shortest simple path (no cycles) from \(s\) to \(u\).

Of course every simple path has at most \(n - 1\) edges

So what if \(D[u]\) improves?

There is a negative cycle!

**A MORE DETAILED IMPLEMENTATION**

With early stopping and checking for negative cycles.