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 $u \neq s$, output a path $s \rightarrow u$ with the smallest total weight (among all paths $s \rightarrow u$).
  i.e., each path $P$ should minimize $w(P) = \sum_{e \in P} w(e)$.

Key insight: after relaxing all, the smallest optimal distance is now optimal.

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
- Can be defined for undirected $G$...

Game AI: path finding with waypoints
1. Divide game world into linear paths, then send game characters straight lines between waypoints.
2. Some linear paths are much faster (lower cost).

Problem: shortest path $s \rightarrow v$ in graph $G$:
1. Let's study directed $G$.
2. Can also be defined for undirected $G$...
3. Otherwise use BFS to find shortest sequence of waypoints (with fewest waypoints).

SINGLE SOURCE SHORTEST PATH
- Input: graph
- Output: shortest path
- Algorithm: Dijkstra’s

CS 341: ALGORITHMS
Lecture 14: graph algorithms V – single source shortest path
Readings: see website
Trevor Brown
https://student.cs.uwaterloo.ca/~cs341
trevor.brown@uwaterloo.ca
**Proof**

**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 (case \( \leq \))
- \( dist[u] \geq \) the total weight of the shortest \( s \rightarrow u \) path (case \( \geq \))

- **CASE \( \leq \)**
  - Let \( P \) be any arbitrary \( s \rightarrow u \) path \( v_0 \rightarrow v_1 \rightarrow \cdots \rightarrow v_\ell \) where \( v_0 = s \) and \( v_\ell = u \)
  - For any index \( j \) let \( L_j \) denote \( w(v_j \rightarrow v_{j-1} \rightarrow v_{j+1}) \)
  - We prove by induction: \( dist[v_j] \leq L_j \) for all \( j \)

- **CASE \( \geq \)**
  - Let \( P' \) be the path \( s \rightarrow \cdots \rightarrow \text{pred}[\text{pred}[u]] \rightarrow \text{pred}[u] \rightarrow u \)
    - i.e., the reverse of following \( \text{pred} \) pointers from \( u \) back to \( s \)
  - We show \( dist[u] \) is as long as this path (and hence as long as the shortest path)
  - Denote the nodes in \( P' \) by \( v_0, v_1, \ldots, v_\ell \) where \( v_0 = s \) and \( v_\ell = u \)
  - Let \( L_j = w(v_j \rightarrow v_{j-1} \rightarrow \cdots \rightarrow v_{j+1}) \)
  - **Proof by induction:** \( v_{j+1} : dist[v_j] = L_j \)
  - Base case: \( dist[v_0] = dist[s] = 0 = L_0 \)
CASE ≥

\[ P' = v_0 \rightarrow \cdots \rightarrow v_k = s \rightarrow \cdots \rightarrow \text{pred[pred[u]]} \rightarrow \text{pred[u]} \rightarrow u \]

\[ L_j = w(v_0 \rightarrow v_1 \rightarrow \cdots \rightarrow v_j) \]

**Inductive step:** Suppose \( \forall_{j > 0} \): \( \text{dist}[v_{j-1}] = L_{j-1} \)

When we set \( \text{pred}[v_j] = v_{j-1} \), we set \( \text{dist}[v_j] = \text{dist}[v_{j-1}] + w(v_{j-1}, v_j) \)

Recall:

\[ \text{dist}(v, u) = \text{dist}(v) \]

\[ \text{dist}(v) = \text{dist}(u) + w(u, v) \]

\[ \text{dist}(v) = \text{dist}(u) \]

By I.H., \( \text{dist}(v_{j-1}) = L_{j-1} + w(v_{j-1}, v_j) \)

By definition \( L_j = L_{j-1} + w(v_{j-1}, v_j) \)

So \( \text{dist}[v_j] = L_j \)

That means it is equal to the length of the shortest path.

**OUTPUTTING ACTUAL SHORTEST PATH(S)?**

To compute the actual shortest path \( s \rightarrow t \)

- Inspect \( \text{pred}[t] \)
  - If it is NULL, there is no such path
  - Otherwise, follow \( \text{pred} \) 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

**RUNTIME**

Each node enqueued and dequeueMin’d once

\( O(n \log n) \)

For each dequeueMin, do \( O(\log n) \) per neighbour

\( O(\log n) \) for each edge

\( O(m \log n) \)

w/adjacency lists

Total time \( O((n + m) \log n) \)

**AN ALTERNATIVE IMPLEMENTATION**

Instead of using a priority queue

Find the minimum \( \text{dist}[] \) node to add to \( \text{OPT} \) via linear search

**Runtime?**

\( O(n^2) \)

Better or worse than \( O((n + m) \log n) ? \)

**SPACE COMPLEXITY?**

\( O(n + m) \)

AN ALTERNATIVE IMPLEMENTATION

Instead of using a priority queue

Find the minimum \( \text{dist}[] \) node to add to \( \text{OPT} \) via linear search

**Runtime?**

\( O(n^2) \)

Better or worse than \( O((n + m) \log n) ? \)
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 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**

If technically suffices to do one iteration of the outer loop.

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

**WORST CASE EXECUTION**

Need \( n \) iterations of outer loop.

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

Since the longest possible path without a cycle can be \( n - 1 \) edges, the edges must be scanned \( n - 1 \) times to ensure the shortest path has been found for all nodes.

**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 \( D[u] = \infty \), 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 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 we do another iteration, and \( D[u] \) improves?

There is a negative cycle!

**BELLMAN-FORD**

The Bellman-Ford algorithm solves the single source shortest path problem in any directed graph without negative weight cycles.

The algorithm is very simple to describe:

Repeat \( n - 1 \) times: relax every edge in the graph (where relax is the updating step in Dijkstra's algorithm).

**A MORE DETAILED IMPLEMENTATION**

With early stopping and checking for negative cycles.

\[
\begin{align*}
\text{BellmanFordCheck}(n, E, \{s\}, s) \\
\text{pred}(s) = \text{new array filled with null} \\
D[s] = \text{new array filled with infinity} \\
D[s] = 0 \\
\text{for } i = 0 \text{ to } n-1 \\
\text{for } (v,w) \in E \\
\text{if } D[w] > D[v] + w \\
\text{pred}(v) = u \\
D[v] = D[w] + w \\
\text{return } \{s\} \\
\end{align*}
\]
BONUS SLIDE

- Why can't you just modify a graph with negative weights
  by: finding the minimum edge weight Wmin, and adding
  that to each edge, so you no longer have negative edges
  and can run Dijkstra's algorithm?

  Exercise: can you find a graph for which this will cause
  Dijkstra's algorithm to return the wrong answer?

  Solution:
  - Consider a graph with 5 nodes: s, a, b, c, t
  - And edges s→a with weight -10, b→t with weight 10
    s→b weight -1, b→c weight -1, c→t weight -1
  - What happens if you modify this graph as proposed,
    then run Dijkstra's to find the shortest path from s to t?