Algorithms, Design & Analysis

Lecture 17: Single Source Shortest Path

Ifra Abdul Rauf & Mishaal Mustazhar

Information Technology University

March 25, 2025

About Your Fellows

- Hi there! We are **Ifra** and **Mishaal**.
- We are Associate Students at ITU.

Single Source Shortest Path

- For **weighted**, **directed** graphs.
- **Goal** : Find the shortest path from a single source node to all other nodes while considering edge weights.

Applications

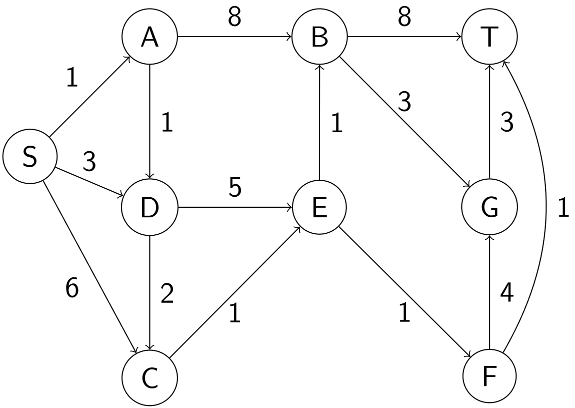
Navigation Systems (e.g., Google Maps, GPS)

- Computes the fastest or shortest route from a given location to multiple destinations.
- Considers road distances, traffic conditions, and time.

Airline Route Optimization

- Determines the most efficient flight paths between airports.
- Reduces fuel costs and optimizes travel time based on available routes.

Example Graph

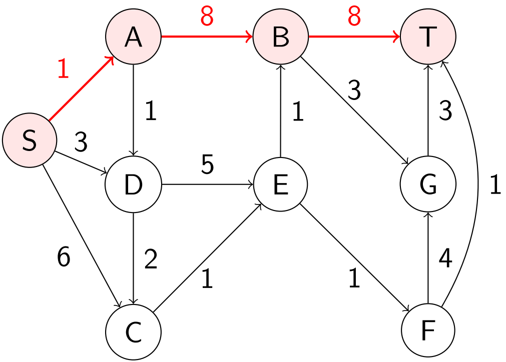


paths from S to T

Let's consider multiple paths from **S** to **T**

Path 1

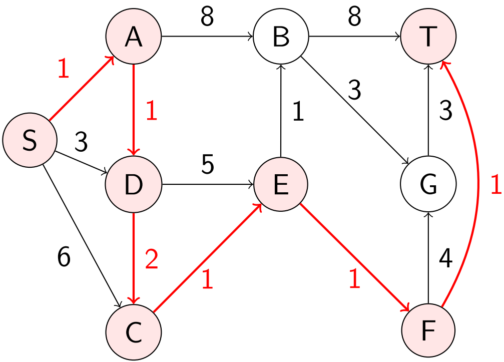
One possible path is:



Path: $S \rightarrow A \rightarrow B \rightarrow T$
Cost: $1 + 8 + 8 = 17$

Path 2

Another possible path is:

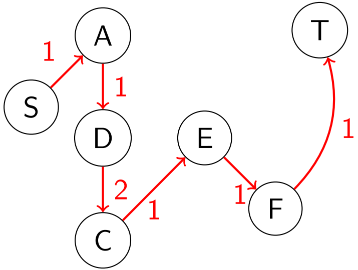


Path: $S \rightarrow A \rightarrow D \rightarrow C \rightarrow E \rightarrow F \rightarrow T$
Cost: $1 + 1 + 2 + 1 + 1 + 1 = 7$.

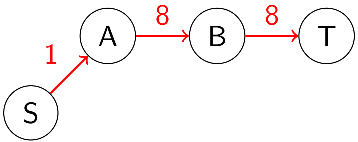
Path Comparison

Comparison of paths

Path 2:



Path 1:



Cost: $1 + 8 + 8 = 17$

Cost: $1 + 1 + 2 + 1 + 1 + 1 = 7$.

Observation: path 2 has more edges, but less cost.

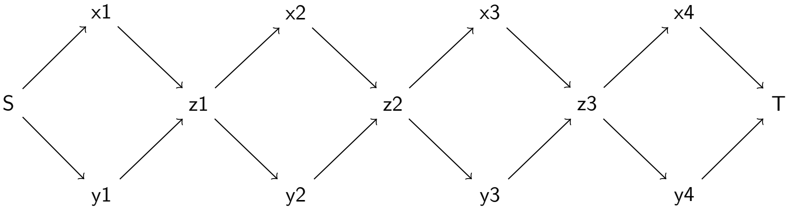
How to Find the Shortest Path?

Naive Approach:

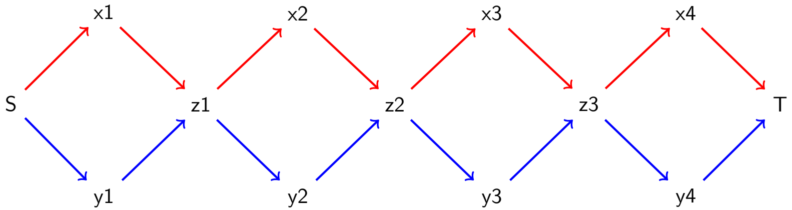
- List all possible paths from the source to the destination.
- Calculate the total weight for each path.
- Choose the one with the minimum weight.

Why Brute Force Doesn't Work?

What happens in graphs with **multiple decision points**?
Consider the following example:



Decision Paths and Their Growth



Each node in the diagram has two possible paths leading to the next stage:

- Red edges (X paths) represent the upper route.
- Blue edges (Y paths) represent the lower route.
- Starting at S, each stage (Z_1, Z_2, Z_3) requires choosing either the X or Y path to continue forward.

Total Unique paths

Since there are **4 decision points** in the example graph, each with **2 choices**:

Total possible paths: $2^4 = 16$**.**

Complex graphs

- Imagine trying to find the **best path** in a complex graph.
- It seems manageable at first... but as the **number of choices increase** at each step, the total **possibilities explode exponentially**!

Exponential Growth in Dense Graphs

Why Brute Force Fails in Dense Graphs?

- If the graph has **100 decision points**, the number of possible paths grows to $\mathbf{2^{100}}$.
- This is an astronomically **large number**. Even the fastest computers can't compute this in reasonable time!
- **Conclusion:** We need efficient algorithms instead of brute force.

Efficient Shortest Path Algorithm

How can we efficiently find the shortest path?

Bellman-Ford Algorithm

Instead of brute force, we use a well-known algorithm like **Bellman-Ford algorithm.**
It applies the concept of **edge relaxation** to find the shortest paths efficiently.

- Iteratively relaxes all edges V - 1 times to compute the shortest paths.
- Time Complexity: O(VE)

Edge Relaxation

Relaxation:
We assign an estimate **d[v]** for the shortest path from S to v.

- Updates the shortest known distance to a node if a shorter path is found.
- Helps **refine** estimates until the shortest path is determined.

Relaxation Pseudocode

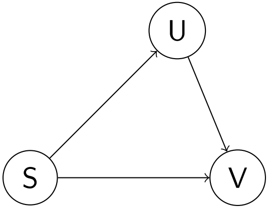
Relax(u, v) :

if $d[u] + w(u, v) < d[v]$ **then**

$d[v] \leftarrow d[u] + w(u, v)$

initial value for d

What should be the initial value for d?



Initial value for d

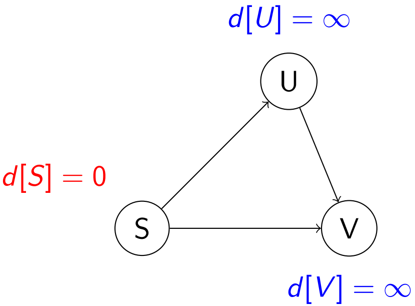
Initial values:

To assign initial values, we start with the simplest possible assumption:

$$d[s] = 0 \quad \text{(The source reaches itself at zero cost)}$$

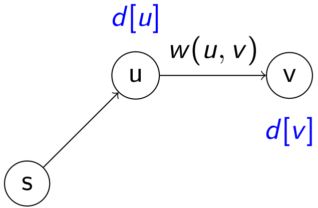
$$d[x] = \infty \quad \text{(for all other nodes, unknown distance)}$$

Value assignment for d



Later, we refine these values using **edge relaxation**.

Relaxing of an edge



Given a directed weighted graph with an edge (u, v) and weight $w(u, v)$, the relaxation step checks if:

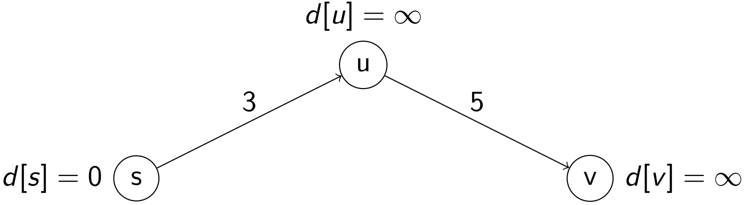
$$d[v] > d[u] + w(u, v)$$

If this condition is **True**, we update:

$$d[v] = d[u] + w(u, v)$$

Example walkthrough

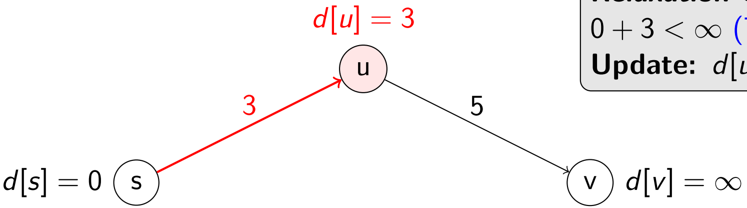
Initial State:



Update d[u]

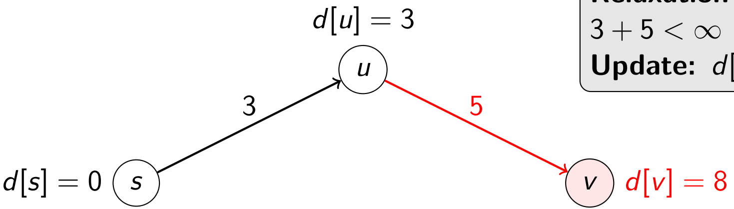
Relaxing Edge: S → U

Relaxation Check:
$0 + 3 < \infty$ (True)
Update: $d[u] = 3$



Update d[v]

Relaxing Edge: U $\rightarrow$ V



Relaxation Check:
$3 + 5 < \infty$ (True)
Update: $d[v] = 8$

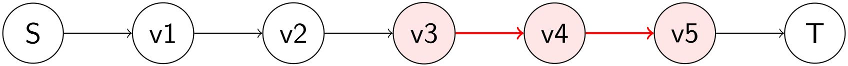
Properties of Shortest Paths

Key properties:

- Optimal Substructure Property
- Edge Relaxation
- Handling Non-Reachable Nodes
- Convergence to True Shortest Path

Optimal Substructure Property

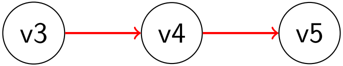
The **shortest path** from S to T is:



What can you say about path from **v3** to **v5**?

EXAMPLE

Since the given graph is the **shortest path** from S to T, then the shortest path from **v3** to **v5** is:



All **sub-paths** of a shortest path are themselves **shortest paths**!

Notation: $\boldsymbol{\delta}$**(s, v)** represent the shortest path from s to v

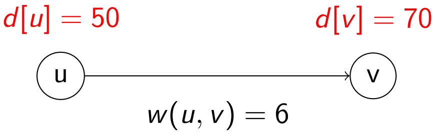
Edge Relaxation Mechanism

Purpose: Refine distance estimates $d[v]$ until they match the true shortest path $\delta(s, v)$.

Relaxation Step:

$$d[v] = \min\left(d[v],\ d[u] + w(u, v)\right)$$

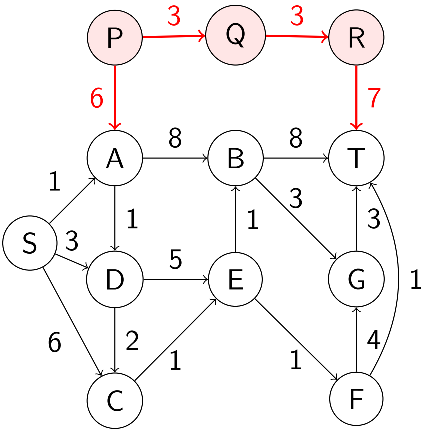
$$70 > 50 + 6 \implies d[v] = 56$$



Unreachable Nodes

What if we have nodes we cannot reach?
How does it affect our path?

Example Graph



If node V is unreachable from S, then:

$$\delta(s, v) = \infty$$

Node P can't be reached from S. hence, no updates during relaxation.

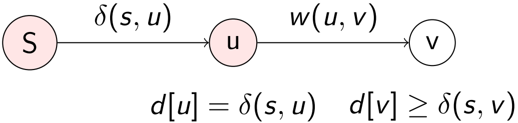
$$\delta(S, P) = \infty$$

$\delta(S, P)$ remains the same.

Perfect Estimate

When do we have a perfect estimate?

- When $d[u] = \delta(s, u)$, we have a perfect estimate for u
- Once perfect, the estimate never changes
- Estimates can either stay the same or decrease



Convergence to True Values

If we relax all the edges in the correct order we get the shortest path

$$S \xrightarrow{\delta(S, v1)} v1 \xrightarrow{\delta(v1, v2)} v2 \xrightarrow{\delta(v2, v3)} v3 \xrightarrow{\delta(v3, v4)} v4 \xrightarrow{\delta(v4, T)} T$$

- **Relax (S,v1):** The shortest path from S to v1 is updated.
- **Relax (v1,v2):** Now that we know $d[v1]$, we update $d[v2]$.
- **Relax (v2,v3):** Using $d[v2]$, we get a better estimate for $d[v3]$.
- Continue this process... **Each step updates the shortest known distance for the next node.**

Final result: After sufficient relaxations, we get the shortest path:

$$\boldsymbol{d[v] = \delta(S, T).}$$

sufficient relaxes

When will we have sufficient relaxes

- A shortest path has at most $V - 1$ edges.
- Bellman-Ford relaxes all edges up to $V - 1$ times.
- After $V - 1$ relaxations, all shortest paths are computed.

Algorithm Overview

Bellman's approach:

for $i = 1$ to $V - 1$ edges (u, v) in G
    Relax every edge (u, v)

Why $V - 1$ iterations?

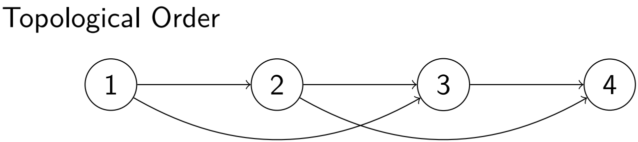
- Longest possible path without cycles has $V - 1$ edges
- Ensures relaxation propagates through the entire graph

Special Case: Directed Acyclic Graphs (DAG)

For DAGs:

- we assume that there are no negative edges
- Perform topological sort ($O(V + E)$)
- Process nodes in topological order
- Relax all outgoing edges from each node

Complexity: $O(V + E)$



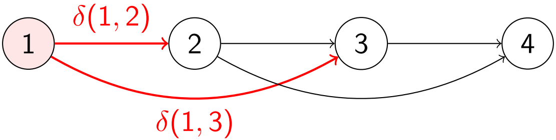
example walk through

First iteration

We relax all the edges going from **1**

- relax(1,2)
- relax(1,3)

Topological Order

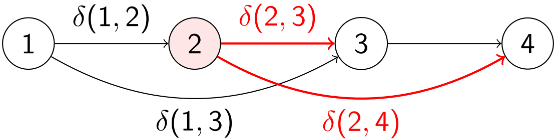


Second iteration

Now we relax all the edges going from **2**

- relax(2,3)
- relax (2,4)

Topological Order

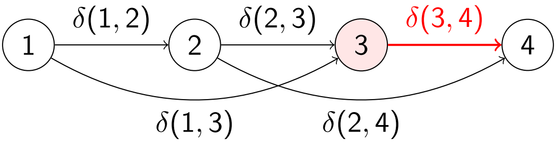


Third iteration

Now we relax all the edges going from **3**

- relax(3,4)

Topological Order



After **n-1** iterations we have relaxed all the edges

Summary of Properties

Summary:

- **Optimal Substructure:** Shortest paths are built from shortest sub-paths.
- **Initialization:** $d[s] = 0, d[v] = \infty$ for others.
- **Relaxation:** Iteratively improves estimates using edges.
- **Non-Reachable Nodes:** Remain at ∞.
- **Convergence:** Guaranteed after $|V| - 1$ relaxations in a graph without negative cycles.

Complexity Analysis

General Case:

- time complexity: $O(VE)$

For dense graph:

- time complexity: $O(V^3)$

Complexity in dense graph

Why $\mathbf{O(V^3)}$ for Dense Graphs?

For dense graphs:

$$|E| \approx |V|^2$$

$$O(VE) = O(V \cdot V^2) = O(V^3)$$

Comparison

Comparison:

- Brute Force: $O(2^n)$ (exponential)
- Bellman-Ford: $O(VE)$ (linear)