

## Lecture 14: Single-Source Shortest Path Algorithms

### Course Learning Outcome

- Use fundamental **graph algorithms**, like traversal, **shortest path** and spanning tree **in the solution of real-life problems**

**Dr. Khalil Yousef**

Adopted from the Slides of the ECE 608 Computational Models and Methods Course at Purdue University

Read Chapter 25 of *Introduction to Algorithms*

## The Shortest-Paths Problem

---

A **shortest-paths problem**, given a weighted, directed graph  $G = (V, E)$  with weight function  $w : E \rightarrow \mathbf{R}$ , is to find the shortest path between two arbitrary vertices,  $u$  and  $v$ .

### Examples:

Determine the shortest route from Aqaba to Irbid.

Determine the shortest distance between two intersections from a street map.

This problem is a generalization of breadth-first search to handle weighted graphs. In BFS, the weight of each edge is 1.

Edge weights can be interpreted, instead of as distances, as time, cost, penalties, etc.

## Shortest-Path Terminology

---

The **weight** of a path  $p = \langle v_0, v_1, \dots, v_k \rangle$  is the sum of the weights of the member edges:

$$w(p) = \sum_{i=1}^k w(v_{i-1}, v_i)$$

The **shortest-path weight** from  $u$  to  $v$  is defined as:

$$\delta(u, v) = \begin{cases} \min\{w(p) : u \xrightarrow{p} v\} & \text{if there is a path from } u \text{ to } v \\ \infty & \text{otherwise.} \end{cases}$$

A **shortest path** from  $u$  to  $v$  is any path  $p$  with  $w(p) = \delta(u, v)$ .

# The Single-Source Shortest-Paths Problem and Variations

---

This lecture focuses on the **single-source shortest-paths problem**: given a graph  $G = (V, E)$ , find the shortest path from a given **source vertex**  $s \in V$  to every other vertex,  $v \in V$ . There are many variations that can be solved with the algorithm for this problem:

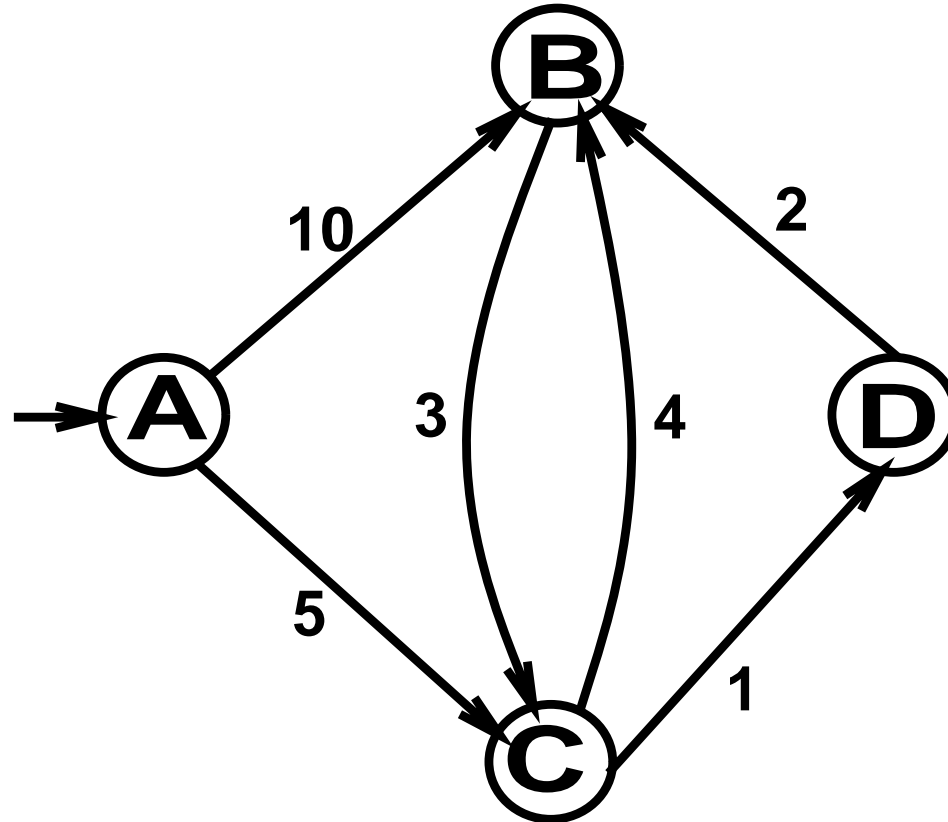
**Single-destination shortest-paths problem**: Find the shortest path to a given **destination** vertex  $t$  from any vertex  $v$ . (Compute  $G^T$  and use the single-source shortest path algorithm with  $t$  as source.)

**Single-pair shortest-path problem**: Find a shortest path from  $u$  to  $v$  only. (No algorithms for this problem are known that run asymptotically faster than the best single-source algorithm in the worst case.)

**All-pairs shortest-paths problems**: Find the shortest path from  $u$  to  $v$  for every pair of vertices  $u$  and  $v$ . (Can use the single-source algorithm for each vertex, but it is best handled as an all-pairs problem.)

## A Single-Source Shortest-Paths Example

---



## A Representation for the Shortest Paths

---

We typically want to compute, not only the weight of each shortest path, but also the vertices in each shortest path.

As in the BFS algorithm, the shortest-paths algorithms will, for a given graph  $G = (V, E)$ , maintain for each vertex  $v \in V$ , a **predecessor**,  $\pi[v]$  (has a value of some  $u \in V$  or  $NIL$ ), so that  $\text{PRINT-PATH}(G, s, v)$  can be used to print the shortest path from  $s$  to  $v$ .

The shortest-paths algorithm will use a predecessor subgraph,  $G_\pi = (V_\pi, E_\pi)$ , induced by the  $\pi$  values, such that:

$$V_\pi = \{v \in V : \pi[v] \neq NIL\} \cup \{s\}$$

$$E_\pi = \{(\pi[v], v) \in E : v \in V_\pi - \{s\}\}$$

We shall prove that the  $\pi$  values produced by the shortest-paths algorithms when they terminate have the property that  $G_\pi$  is a **shortest-paths tree**, a tree rooted at  $s$  containing a shortest path from  $s$  to each vertex  $v \in V$  reachable from  $s$ .

## Shortest-Paths Tree

---

Let  $G = (V, E)$  be a weighted, directed graph with weight function  $w : E \rightarrow \mathbf{R}$  with no negative-weight cycles reachable from  $s \in V$ , then a **shortest-paths tree** rooted at  $s$  is defined formally as a subgraph  $G' = (V', E')$ , where  $V' \subseteq V, E' \subseteq E$  such that:

1.  $V'$  is the set of vertices reachable from  $s$  in  $G$ .
2.  $G'$  forms a rooted tree with root  $s$ .
3.  $\forall v \in V'$ , the unique simple path from  $s$  to  $v$  in  $G'$  is a shortest path from  $s$  to  $v$  in  $G$ .

Because shortest paths are not necessarily unique, neither is a shortest-paths tree always unique.

## Optimal Substructure of a Shortest Path

---

The shortest-paths algorithms exploit the property that subpaths of a shortest path are shortest paths.

**Lemma 25.1. (Subpaths of shortest paths are shortest paths)**

The proof is provided in the book



## Shortest-Paths Algorithms: Initialization

---

The single-source shortest-paths algorithms operate on a weighted, directed graph  $G = (V, E)$  and use two arrays,  $\pi$  and  $d$ , to calculate the shortest paths from  $s$  to each  $v \in V$ . When the algorithms terminate, for  $v \in V$ ,  $\pi[v]$  is the predecessor of  $v$  in the shortest path from  $s$  to  $v$  and  $d[v]$  is the weight of that path.

The following routine is used by the single-source shortest-paths algorithms to initialize the arrays:

INITIALIZE-SINGLE-SOURCE( $G, s$ )

1. **for** each vertex  $v \in V[G]$
2.     **do**  $d[v] \leftarrow \infty$
3.          $\pi[v] \leftarrow \text{NIL}$
4.  $d[s] \leftarrow 0$

## Shortest-Paths Algorithms: Relaxation

---

The shortest-paths algorithms use a technique called **relaxation**. The basic idea is that  $d[v]$  is an upper bound on the weight of a shortest path from source  $s$  to  $v$ , and as such is called a **shortest-path estimate**, which is reduced from  $\infty$  until it finally reaches the shortest-path weight value  $\delta(s, u)$ .

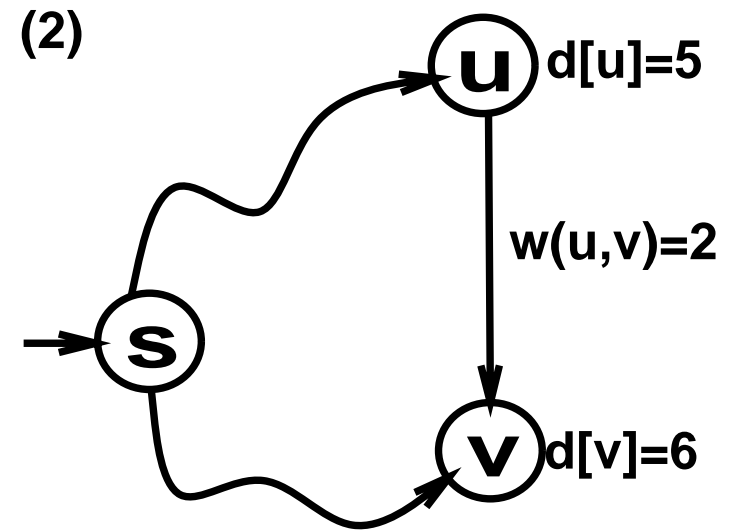
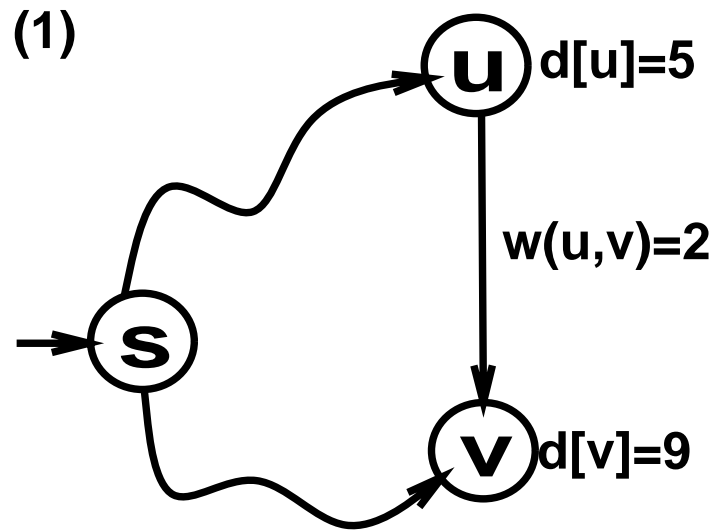
RELAX( $u, v, w$ ) updates  $d[v]$  (and  $\pi[v]$ ) by examining the impact of the weight of edge  $(u, v)$ .

```
RELAX( $u, v, w$ )  
1. if  $d[v] > d[u] + w(u, v)$   
2.   then  $d[v] \leftarrow d[u] + w(u, v)$   
3.      $\pi[v] \leftarrow u$ 
```

Relaxation is the only means by which shortest-path estimates and predecessors change once they are initialized in all of the single-source shortest-path algorithms.

## Shortest-Paths Algorithms: Relaxation

---



## Properties of Relaxation

---

**Lemma 25.4.** Let  $G = (V, E)$  be a weighted, directed graph with weight function  $w : E \rightarrow \mathbf{R}$ , and let  $(u, v) \in E$ . Then, immediately after relaxing  $(u, v)$  using  $\text{RELAX}(u, v, w)$ ,  $d[v] \leq d[u] + w(u, v)$ .

**Proof:**

If  $d[v] > d[u] + w(u, v)$  just prior to the call  $\text{RELAX}(u, v, w)$ , then  $d[v] = d[u] + w(u, v)$  afterward. On the other hand, if  $d[v] \leq d[u] + w(u, v)$  just prior to the call, then nothing changes.

□

**Lemma 25.5.** Let  $G = (V, E)$  be a weighted, directed graph with weight function  $w : E \rightarrow \mathbf{R}$ , let  $s$  be the source vertex, and let the graph be initialized using  $\text{INITIALIZE-SINGLE-SOURCE}(G, s)$ , then  $d[v] \geq \delta(s, v)$  for all  $v \in V$ , and this invariant is maintained over any sequence of relaxation steps on  $E$ . Once  $d[v]$  achieves its lower bound  $\delta(s, v)$ , it never changes.

## Dijkstra's Algorithm

---

- Dijkstra's Algorithm uses a greedy strategy together with the assumption of no negative edge weights to determine the shortest paths from a source vertex  $s$  in a weighted, directed graph  $G = (V, E)$ , represented using adjacency lists, to all  $v \in V$ .
- The algorithm creates a set  $S$  of vertices whose shortest-path weights from  $s$  have been determined.
- The algorithm repeatedly selects the minimum vertex from a priority queue  $Q$ , containing vertices in  $V - S$  keyed by their  $d$  values.
- It is like BFS except that it uses a priority queue, keyed on  $d$ . It is also similar to MST-PRIM.

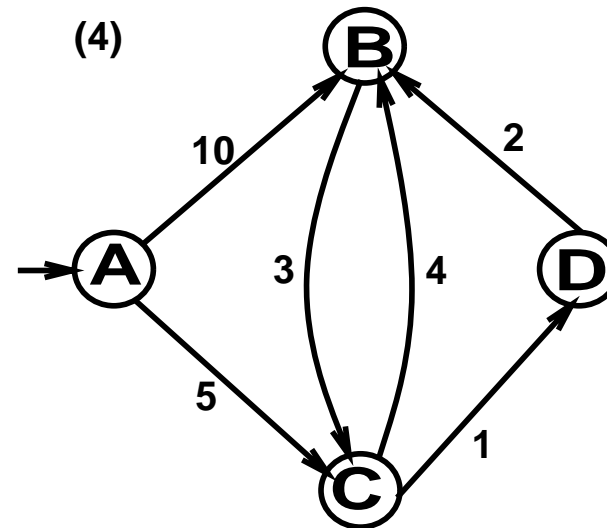
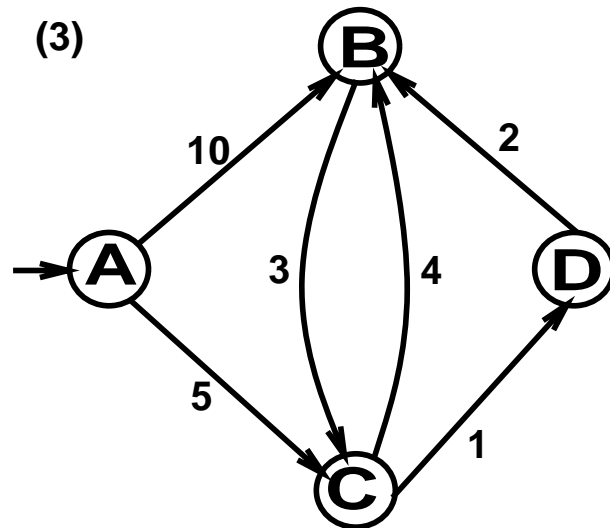
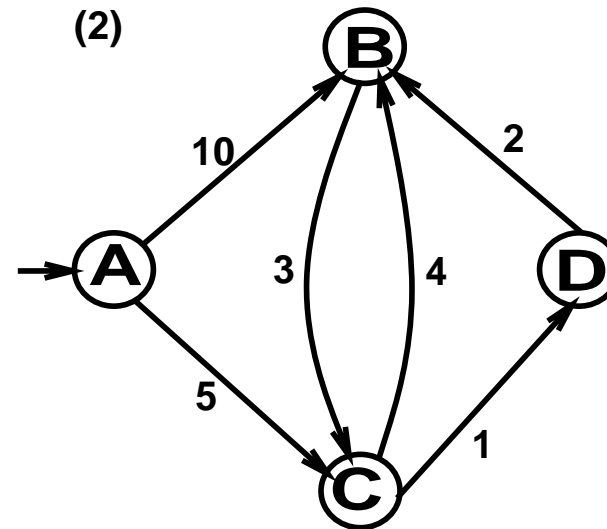
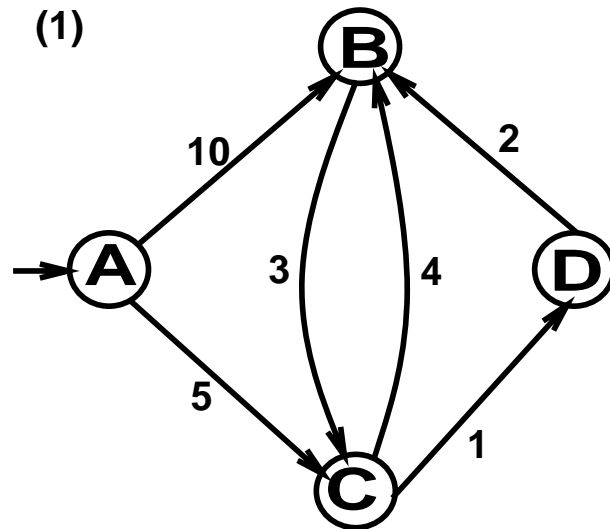
## DIJKSTRA( $G, w, s$ )

---

```
DIJKSTRA( $G, w, s$ )
1. INITIALIZE-SINGLE-SOURCE( $G, s$ )
2.  $S \leftarrow \emptyset$ 
3.  $Q \leftarrow V[G]$ 
4. while  $Q \neq \emptyset$ 
5.     do  $u \leftarrow \text{EXTRACT-MIN}(Q)$ 
6.          $S \leftarrow S \cup \{u\}$ 
7.         for each vertex  $v \in \text{Adj}[u]$ 
8.             RELAX( $u, v, w$ )
```

## DIJKSTRA Example

---



## Running Time of Dijkstra's Algorithm

---

EXTRACT-MIN:  $|V|$  times.

DECREASE-KEY:  $|E|$  times.

Hence, the worst case running time of DIJKSTRA can be characterized by the equation  $|V| T_{\text{EXTRACT-MIN}} + |E| T_{\text{DECREASE-KEY}}$ .

$Q$	$T_{\text{EXTRACT-MIN}}$	$T_{\text{DECREASE-KEY}}$	Total
Array	$O(V)$	$O(1)$	$O(V^2)$
Binary Heap	$O(\lg V)$	$O(\lg V)$	$O((E + V) \lg V)$
Fibonacci Heap	$O(\lg V)$	$O(1)$	$O(V \lg V + E)$



## The Bellman-Ford Algorithm

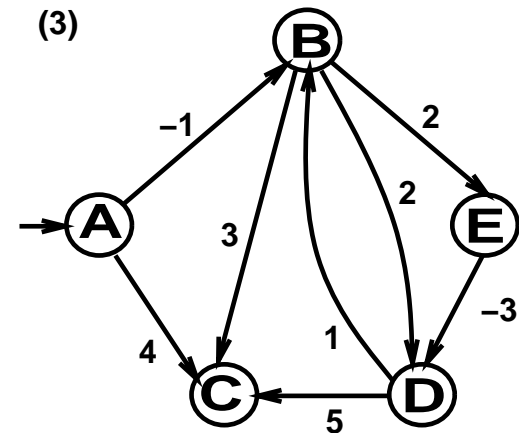
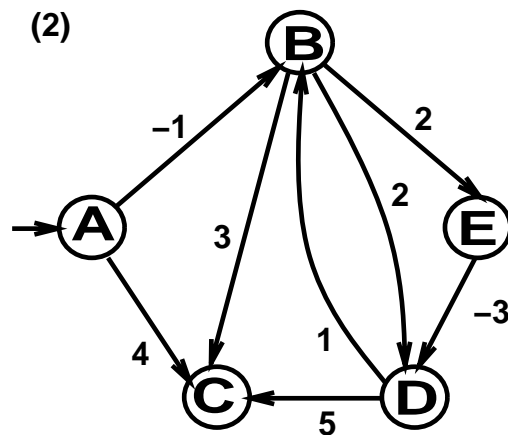
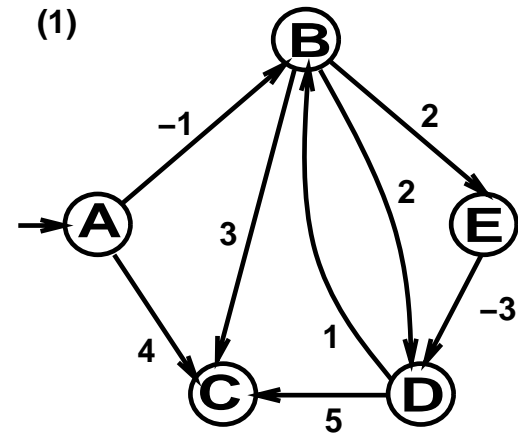
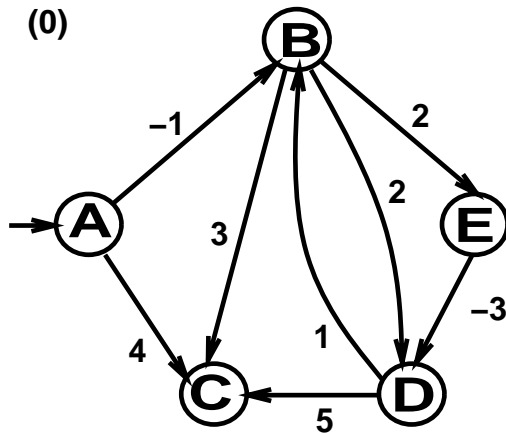
---

BELLMAN-FORD( $G, w, s$ ) is a single-source shortest-paths algorithm that supports negative edge weights. Given a weighted, directed graph  $G = (V, E)$  with source  $s$  and weight function  $w : E \rightarrow \mathbf{R}$ , it returns a solution if and only if there is no negative-weight cycle reachable from  $s$  (a boolean value of FALSE indicates that no solution exists because there is a negative-weight cycle).

```
BELLMAN-FORD( $G, w, s$ )
1. INITIALIZE-SINGLE-SOURCE( $G, s$ )
2. for  $i \leftarrow 1$  to  $|V[G]| - 1$ 
3.     do for each edge  $(u, v) \in E[G]$ 
4.         do RELAX( $u, v, w$ )
5. for each edge  $(u, v) \in E[G]$ 
6.     do if  $d[v] > d[u] + w(u, v)$ 
7.         then return FALSE
8. return TRUE
```

## BELLMAN-FORD Example

Consider the edges in the following order:  $(A,B)$ ,  $(A,C)$ ,  $(B,C)$ ,  $(B,D)$ ,  $(D,B)$ ,  $(D,C)$ ,  $(E,D)$ ,  $(B,E)$ .



## The Running Time of BELLMAN-FORD

---

INITIALIZE-SINGLE-SOURCE takes  $\Theta(V)$  time.

In lines 2-4, there are  $|V| - 1$  passes over  $E$  edges, which takes  $O(V E)$  time.

Lines 5-7 takes  $O(E)$  time.

Hence, the running time of BELLMAN-FORD is  $O(V E)$ .