# COMPSCI 311: Introduction to Algorithms

Lecture 8: Shortest Paths

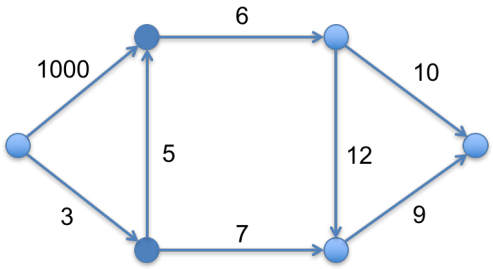
Marius Minea

University of Massachusetts Amherst

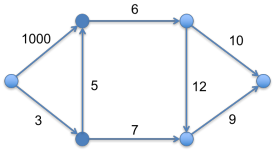
slides credit: Dan Sheldon

## Shortest Paths Problem

**Problem**: find shortest paths in a directed graph with edge *lengths* (e.g., Google maps)




## Let's Formalize the Problem




- Directed graph $G = (V, E)$ with **nonnegative** edge *lengths* $\ell(e) \geq 0$
- Define *length* of path $P$ consisting of edges $e_1, e_2, \ldots, e_k$ as

$$\ell(P) = \ell(e_1) + \ell(e_2) + \ldots + \ell(e_k)$$

- Starting node $s$
- Let $d(v)$ be the length of shortest $s \rightsquigarrow v$ path.
- **Problem:** Can we efficiently find $d(v)$ for all nodes $v \in V$?
- **Question:** Why for all nodes at the same time?

## Clicker Question 1

Consider a car's GPS navigation system. What shortest paths does it compute at any given time?

A. Single sink: from every node to one node $t$.

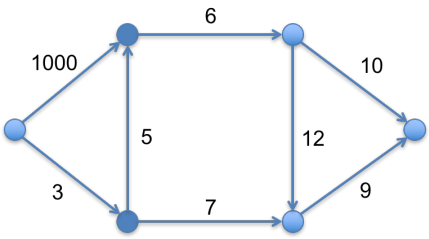
B. Source-sink: from one node $s$ to another node $t$.

C. Single source: from one node $s$ to every other node.

D. All pairs: between all pairs of nodes.

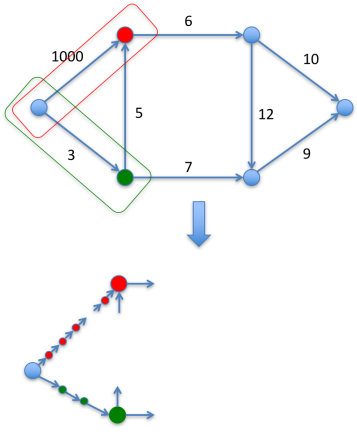
## Shortest Paths Problem

Suppose all edges have integer length.
Can we use BFS to solve this problem?

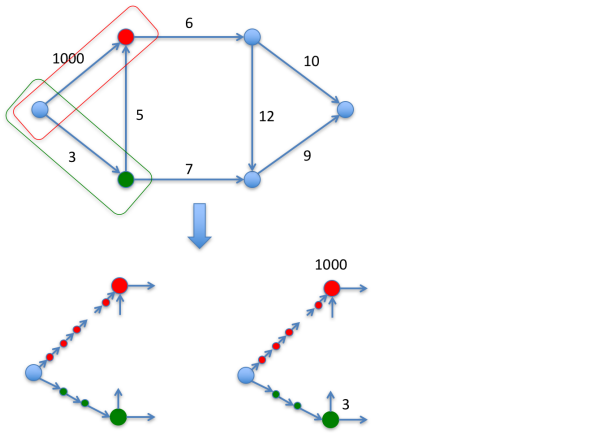



Recall: nodes in layer $L_i$ are at distance $i$ from start.

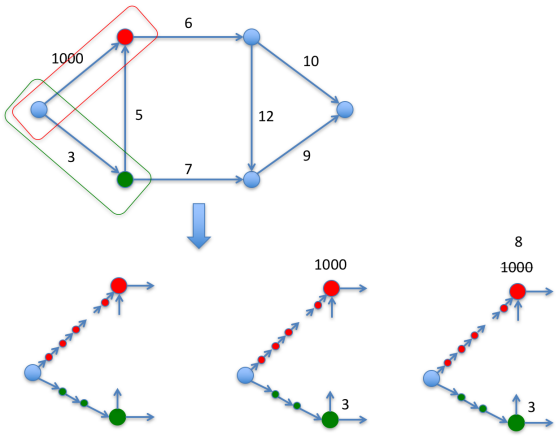
## Shortest Paths Problem



## Shortest Paths Problem




## Shortest Paths Problem




## Clicker Question 2

If edge lengths are integers, and $C$ is the maximum length, the running time is (choose the most restrictive):

A. $O(C + m + n)$

B. $O(n + C \cdot m)$

C. $O(m + C \cdot n)$

D. $O(C \cdot (m + n))$

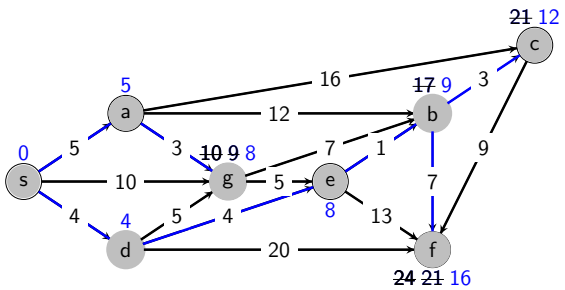
Hint: Why is BFS $O(m + n)$ and not $O(m)$ ?

## Computing Shortest Paths

Idea: keep track of expanding "wavefront" (arrival time at $v = d(v)$)

- find next node $v$ to be hit by wave
  ($=$ unexplored node closest to $s$)
- explore $v$: update tentative arrival time at each neighbor

## Shortest Path Example




## Towards an Algorithm

Notation:

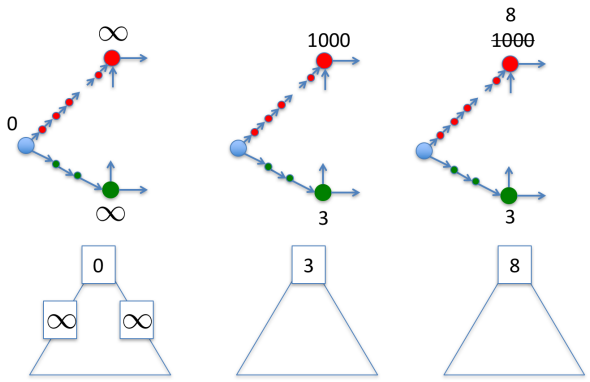
- $d'(v)$ — earliest tentative arrival time so far for node $v$
- $d(v)$ — shortest distance (actual arrival time)

How to keep track of the wavefront?

- Find *next arrival*: node $v$ with smallest $d'(v)$
- Set shortest distance: $d(v) = d'(v)$
- Update $d'(v)$ for neighbors of $v$ if path through $v$ shorter

What data structure supports find smallest and update values?
Priority queue.

## Shortest Paths Problem




## Dijkstra's Algorithm

set $A = V$ ▷ Priority queue
set $d'(v) = \infty$ for all nodes ▷ Tentative arrival time
set $d'(s) = 0$
**while** $A$ not empty **do** ▷ Nodes left to explore
  extract node $v \in A$ with smallest $d'(v)$ value
  set $d(v) = d'(v)$ ▷ Wave arrives at $v$
  **for** all edges $(v, w)$ where $w \in A$ **do**
    **if** $d(v) + \ell(v, w) < d'(w)$ **then** ▷ Shorter path??
      $d'(w) = d(v) + \ell(v, w)$
    **end if**
  **end for**
**end while**

## Running Time?

Use heap-based priority queue for $A$

set $A = V$
set $d'(v) = \infty$ for all nodes
set $d'(s) = 0$
**while** $A$ not empty **do**
  extract node $v \in A$ with smallest $d'(v)$ value ▷ Extract-min
  set $d(v) = d'(v)$
  **for** all edges $(v, w)$ where $w \in A$ **do**
    **if** $d(v) + \ell(v, w) < d'(w)$ **then**
      $d'(w) = d(v) + \ell(v, w)$ ▷ Update-key
    **end if**
  **end for**
**end while**

- $n$ extract-min operations. $O(n \log n)$
- $m$ update-key operations. $O(m \log n)$
- **Total**: $O((m + n) \log n)$

## Tracking the Shortest Path

Keep track of node that last updated arrival time $d'(v)$
Call it prev($v$) = predecessor in shortest path

set $A = V$
set prev($v$) = null for all nodes
set $d'(v) = \infty$ for all nodes
set $d'(s) = 0$
**while** $A$ not empty **do**
  extract node $v \in A$ with smallest $d'(v)$ value
  set $d(v) = d'(v)$
  **for** all edges $(v, w)$ where $w \in A$ **do**
    **if** $d(v) + \ell(v, w) < d'(w)$ **then**
      $d'(w) = d(v) + \ell(v, w)$
      prev($w$) = $v$
    **end if**
  **end for**
**end while**

## Proof of Correctness

- Note (**optimal substructure**): If $u$ is a node on a shortest path $s \rightsquigarrow u \rightsquigarrow v$, then $s \rightsquigarrow u$ is also a shortest path.
- **Idea**: nodes are explored in increasing order of distance from $s$.
- At each step, we have:
  $A$: set of nodes still to explore
  $S = V \setminus A$: explored nodes, $d(v)$ assigned
- **Claim (invariant)**:
  1. for $v \in S$, $d(v)$ is length of shortest $s \rightsquigarrow v$ path
  2. for $v \in A$, $d'(v)$ is the length of the shortest $s \rightsquigarrow v$ path with all nodes in $S$ except $v$.
- Proof: By induction on $|S|$

## Induction Proof

Base case: Initially $S = \emptyset$. (1) is vacuously true.
The only path satisfying (2) is the empty path, and $d'(s) = 0$. ✓

Induction step:

- Assume the invariant is true for $|S| = k \geq 0$.
- Let $v$ = next node added to $S$, with $d'(v) = \min_{w \in A} d'(w)$.
  We claim (1) $d(v) = d'(v)$ is the shortest path.
- By (2), $d'(v)$ is shortest path with all prior nodes in $S$.

## Induction Proof (2)

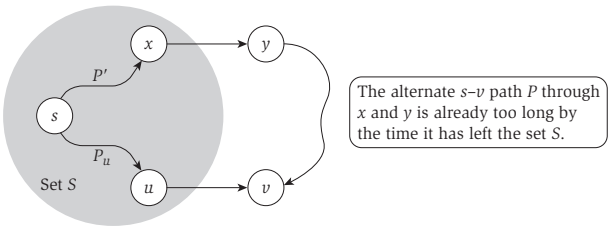



**Figure 4.8** The shortest path $P_v$ and an alternate $s$-$v$ path $P$ through the node $y$. (Kleinberg & Tardos)

- Consider a path with some prior node not in $S$.
  Let $y \in A$ be the first such node on a path $s \rightsquigarrow y \rightsquigarrow v$.
  But then $\ell(s \rightsquigarrow y \rightsquigarrow v) \geq \ell(s \rightsquigarrow y) \geq d'(y) \geq d'(v)$,
  so we *cannot* have a shorter path.
- Critical point: segment $y \rightsquigarrow v$ cannot have **negative** length.

## Proof: Maintaining the Invariant

Show we maintain (2): for $v \in A$, $d'(v)$ is the length of the shortest $s \rightsquigarrow v$ path with all nodes in $S$ except $v$.

**for** all edges $(v, w)$ where $w \in A$ **do**
  **if** $d(v) + \ell(v, w) < d'(w)$ **then**
    $d'(w) = d(v) + \ell(v, w)$
  **end if**
**end for**

Adding $v$ to $S$, we get new such paths $s \rightsquigarrow w$ for all edges $(v, w)$.

Updating $d'(w) = \min(d'(w), d(v) + \ell(v, w))$ maintains invariant.

## Clicker Question #3

Dijkstra's algorithm works for nonnegative edges.
(We'll discuss the Bellman-Ford algorithm, which can handle negative edges.)

In general, there exists a shortest $s \rightsquigarrow v$ path if

A. There are no negative-length edges on any path $s \rightsquigarrow v$

B. There is no negative-length cycle on any path $s \rightsquigarrow v$

C. Any path $s \rightsquigarrow v$ that has a negative-length cycle also has a positive-length cycle

D. Any path $s \rightsquigarrow v$ that has a negative-length cycle also has a positive-length cycle, longer in absolute value

## Dijkstra's algorithm: which priority queue?

Performance. Depends on PQ: $n$ INSERT, $n$ DELETE-MIN, $\leq m$ DECREASE-KEY.

- Array implementation optimal for dense graphs. ⟵ $\Theta(n^2)$ edges
- Binary heap much faster for sparse graphs. ⟵ $\Theta(n)$ edges
- 4-way heap worth the trouble in performance-critical situations.

| priority queue | INSERT | DELETE-MIN | DECREASE-KEY | total |
|---|---|---|---|---|
| unordered array | $O(1)$ | $O(n)$ | $O(1)$ | $O(n^2)$ |
| binary heap | $O(\log n)$ | $O(\log n)$ | $O(\log n)$ | $O(m \log n)$ |
| d-way heap (Johnson 1975) | $O(d \log_d n)$ | $O(d \log_d n)$ | $O(\log_d n)$ | $O(m \log_{m/n} n)$ |
| Fibonacci heap (Fredman–Tarjan 1984) | $O(1)$ | $O(\log n)$ † | $O(1)$ † | $O(m + n \log n)$ |
| integer priority queue (Thorup 2004) | $O(1)$ | $O(\log \log n)$ | $O(1)$ | $O(m + n \log \log n)$ |

† amortized



slide credit: Kevin Wayne / Pearson

## Integers: Special Case

Thorup 1999: Single-source shortest paths in undirected graphs with **positive integer** edge lengths in $O(m)$ time.

Does *not* explore nodes by increasing distance from $s$.

**Undirected Single-Source Shortest Paths with Positive Integer Weights in Linear Time**

MIKKEL THORUP

*AT&T Labs Research, Florham Park, New Jersey*

Abstract. The single-source shortest paths problem (SSSP) is one of the classic problems in algorithmic graph theory: given a positively weighted graph $G$ with a source vertex $s$, find the shortest path from $s$ to all other vertices in the graph.

Since 1959, all theoretical developments in SSSP for general directed and undirected graphs have been based on Dijkstra's algorithm, visiting the vertices in order of increasing distance from $s$. Thus, any implementation of Dijkstra's algorithm sorts the vertices according to their distances from $s$. However, we do not know how to sort in linear time.

Here, a deterministic linear time and linear space algorithm is presented for the undirected single source shortest paths problem with positive integer weights. The algorithm avoids the sorting bottleneck by building a hierarchical bucketing structure, identifying vertex pairs that may be visited in any order.