COMPSCI 311: Introduction to Algorithms

Lecture 7: Shortest Paths

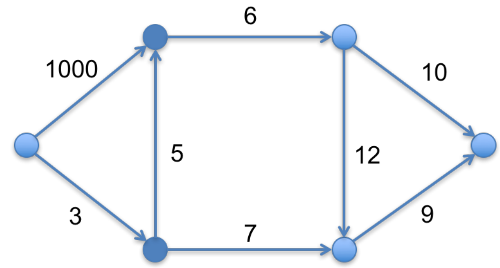
Marius Minea

University of Massachusetts Amherst

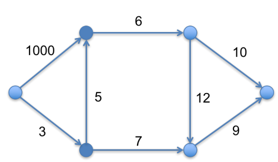
slides credit: Dan Sheldon

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 edge *lengths* $\ell(e) > 0$
- 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?

Shortest paths: quiz 2

Which variant in car GPS?

A. Single source: from one node s to every other node.

B. Single sink: from every node to one node t.

C. Source–sink: from one node s to another node t.

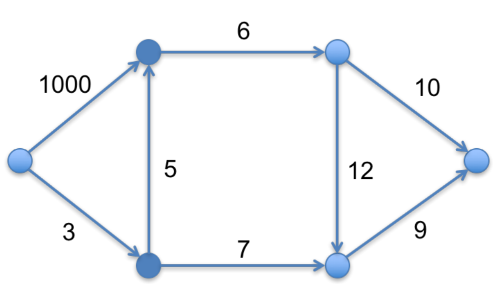
D. All pairs: between all pairs of nodes.

6

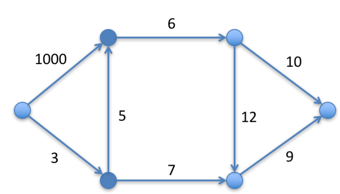
slide credit: Kevin Wayne / Pearson

Shortest Paths Problem

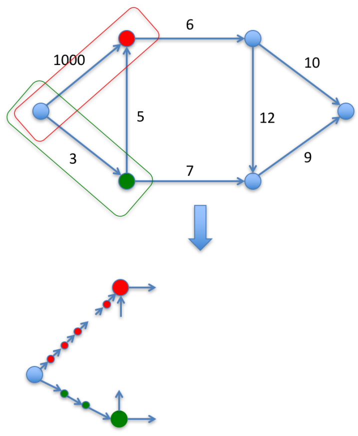
Suppose all edges have integer length.
Can we use BFS to solve this problem?



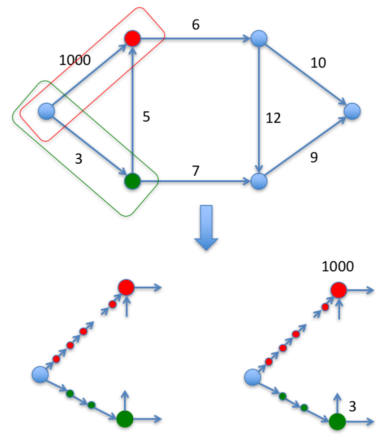
Shortest Paths Problem



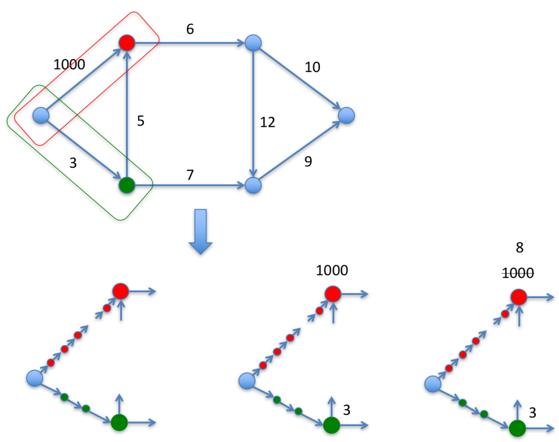
Shortest Paths Problem



Shortest Paths Problem



Shortest Paths Problem



Shortest Paths Problem

Idea: keep track of the "wavefront"

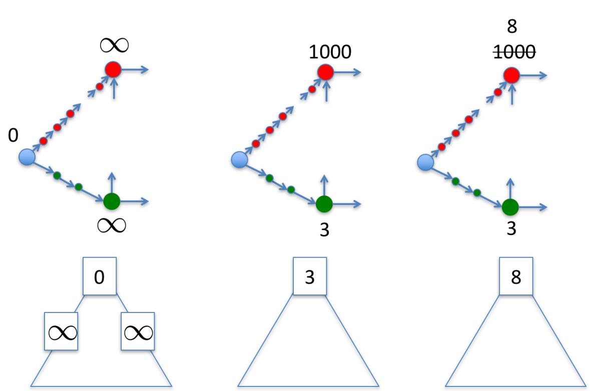
- $d'(v)$ — best tentative arrival time so far for node v
- $d(v)$ — actual arrival time

What's required to keep track of the wavefront?

- Find *next arrival*: node v with smallest $d'(v)$
- Set arrival time: $d(v) = d'(v)$
- Update $d'(v)$ for neighbors of v if they get better "offers"

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** ▷ Better offer?
      $d'(w) = d(v) + \ell(v, w)$
    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**: $O((m + n) \log n)$

Finding the Actual Path

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

Set $A = V$
Set $d'(v) = \infty$ for all nodes
Set prev(v) = null
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

- Let $S = V \setminus A$ be the set of *explored* nodes at any point in the algorithm—those v for which we have assigned $d(v)$
- **Observation**: for $v \notin S$, the value $d'(v)$ is the minimum value $d(u) + \ell(u, v)$ over all edges (u, v) where $u \in S$, $v \notin S$.
  - length of shortest path to v that remains in S until final hop.
- **Claim (invariant)**: for $v \in S$, the value $d(v)$ is the length of the shortest $s \rightsquigarrow v$-path
- Proof: By induction on $|S|$

Proof (by induction)

- *Base case:* Initially $S = \{s\}$ and $d(s) = 0$. ✓
- *Induction step:*
  - Assume the invariant is true after the kth execution of the while loop, when $|S| = k$.
  - Let v be the next node added to S, and let (u, v) be the preceding edge.
    Then $d'(v) = d(u) + \ell(u, v)$, and $d'(v) \leq d'(x)$ for any node $x \notin S$.
  - Let P_u be the shortest $s \rightsquigarrow u$ path, which has length $d(u)$
  - Let $P_v = P_u \cup (u, v)$ be the path found by Dijkstra, which has length $\ell(P_v) = d'(v) = d(u) + \ell(u, v)$

Induction Proof (cont.)

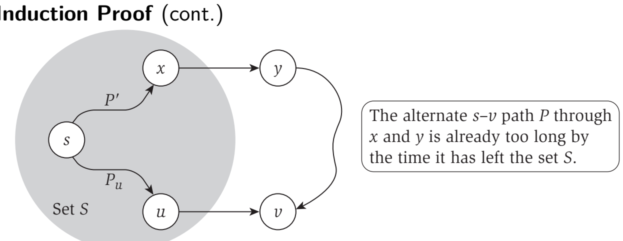


Figure 4.8 The shortest path P_v and an alternate s-v path P through the node y.
(Kleinberg & Tardos)

- *Induction step:* (cont.)
  - Consider any other $s \rightsquigarrow v$ path P. We'll argue that P is *already* longer than P_v by the time it first leaves S.
  - Let (x, y) be the first edge in P with $x \in S, y \notin S$, and let P' be the subpath of P from s to x
  - Then,

$$\ell(P) \geq \ell(P') + \ell(x, y) \geq d(x) + \ell(x, y) \geq d'(y) \geq d'(v) = \ell(P_v)$$

Clicker Question #2

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

Integers: Special Case

Thorup 1999: Solved single-source shortest paths problem 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.

Spanning Trees

Network Design Problem

- **Given**: an undirected graph $G = (V, E)$ with edge costs (weights) $c_e > 0$.
  Assume for now that all edge weights are distinct.
- **Find**: subset of edges $T \subseteq E$ such that (V, T) is connected and the total cost of edges in T is as small as possible
- Call $T \subseteq E$ a *spanning tree* if (V, T) is a tree (*connected*, no cycles)
- **Claim**: in a minimum-cost solution, T is a spanning tree.
- This is the **minimum spanning tree (MST) problem**.

Cuts in Graphs

- A key to understanding MSTs is a concept called a cut.
- **Definition**: A cut in G is a partition of the nodes into two nonempty subsets $(S, V - S)$.
- **Definition**: Edge $e = (v, w)$ crosses cut $(S, V - S)$ if $v \in S$ and $w \in V - S$.
  The cutset of a cut is the set of edges that cross the cut.

Minimum spanning trees: quiz 2

Let C be a cycle and let D be a cutset. How many edges do C and D have in common? Choose the best answer.

A. 0

B. 2

C. not 1

D. an even number