http://www.cs.unh.edu/~ruml/cs758
Shortest Path Problems
single source/destination pair
single source, all destinations: harder?
single destination, all sources
all-pairs

non-uniform weights?
negative weights?
negative-weight cycles?
optimal substructure
triangle inequality: \( \delta(s, v) \leq \delta(s, u) + w(u, v) \)
‘relaxing’: constraint is met
Dijkstra’s Algorithm

A Faster Algorithm

Bellman-Ford
1930–2002; Turing Award ’72 invented RPN, self-stabilizing algorithms, semaphores

The goto statement considered harmful
structured programming (while!), formal verification

“I mean, if 10 years from now, when you are doing something quick and dirty, you suddenly visualize that I am looking over your shoulders and say to yourself ‘Dijkstra would not have liked this,’ well, that would be enough immortality for me.”
The Algorithm

1. for each vertex, v.d ← ∞
2. s.d ← 0
3. Q ← all vertices
4. while Q is not empty
   5. u ← remove vertex in Q with smallest d
   6. for each edge (u, v) from u
      7. if v.d > u.d + w(u, v)
      8. v.d ← u.d + w(u, v)
      9. v.π ← u

correctness?
running time?
negative weights?
Correctness

Key property: popped vertices have $d = \delta$
Proof by induction.

**Base case:** $s$

**Assumption:** previously popped vertices have $d = \delta$

**Inductive Step:** proof by contradiction. Consider freshly popped $v$. Assume its current path is too long. Since $d = \delta$ for all previously popped, then if $v$’s predecessor along the optimal path were previously popped, then $v.d$ would be correct. So there exists an unpopped vertex $u$ along the shortest path. Let $u$ be the first such vertex in the path. Note $u.d = \delta(s, u)$. Since it is earlier on the optimal path, $u.d = \delta(s, u) \leq \delta(s, v) < v.d$. But then we would have popped $u$ instead of $v$: contradiction!
Running Time

$O((V + E) \lg V)$

$O(V \lg V + E)$ using a Fibonacci heap
Dijkstra's Algorithm

- Dijkstra
- Algorithm
- Correctness
- Running Time

Break

A Faster Algorithm

Bellman-Ford

- asst 10
- don’t cheat!
- wildcard vote Nov 16
A Faster Algorithm
1. for each vertex, \( v.d \leftarrow \infty \)
2. \( s.d \leftarrow 0 \)
4. for each vertex \( u \) in topologically sorted order
6. for each neighbor \( v \)
7. if \( v.d > u.d + w(u, v) \)
8. \( v.d \leftarrow u.d + w(u, v) \)
9. \( v.\pi \leftarrow u \)

correctness?
running time?
edge weights?
Bellman-Ford
1. for each vertex, \( v.d \leftarrow \infty \)
2. \( s.d \leftarrow 0 \)
3. repeat \(|V|\) times
4. for each edge \((u, v)\)
5. if \( v.d > u.d + w(u, v) \)
6. \( v.d \leftarrow u.d + w(u, v) \)
7. \( v.\pi \leftarrow u \)

correctness?
running time? how to make it faster?
negative cycles?
Path relaxation: If we relax all the edges along a shortest $u,v$ path in order, then $v.d = \delta(u,v)$, even if other relaxations are performed. Proof: induction on length of path

Bellman-Ford: Proof: Consider a shortest path. Its length will be $\leq |V| - 1$. Each Bellman-Ford iteration considers all edges.
For example:

- What’s still confusing?
- What question didn’t you get to ask today?
- What would you like to hear more about?

Please write down your most pressing question about algorithms and put it in the box on your way out. 

*Thanks!*