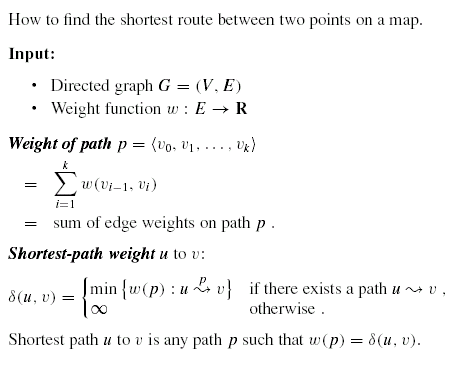
**CS430 Lecture 27 Activities**

Shortest Path Problem



Variants

* Single-source: Find shortest paths from a given source vertex s ∈ V to every vertex v ∈ V.
* Single-destination: Find shortest paths to a given destination vertex.
* Single-pair: Find shortest path from u to v. No way known that’s better in worst case than solving single-source.
* All-pairs: Find shortest path from u to v for all u, v ∈ V. We’ll see algorithms for all-pairs in the next chapter.

Negative-weight edges - OK, as long as no negative-weight cycles are reachable from the source.

* If we have a negative-weight cycle, just keep going around it, and get *w(s, v)*=-∞for all *v* on the cycle.
* But OK if the negative-weight cycle is not reachable from the source.
* Some algorithms work only if there are no negative-weight edges in the graph.

1. What would the brute force approach be to solve the shortest path problem, and what is its run time?

2. Prove optimal substructure for the shortest path problem

Output of single-source shortest-path algorithm

For each vertex *v ∈*  *V*:

* *d*[*v*] = *δ(s, v),* Initially, *d*[*v*]=∞, Reduces as algorithms progress. But always maintain *d*[*v*] ≥ *δ(s, v)*. Call *d*[*v*] a *shortest-path estimate*.
* *π*[*v*] = predecessor of *v* on a shortest path from *s,* If no predecessor, *π*[*v*] = NIL, *π* induces a tree—*shortest-path tree*

Initialization - All the shortest-paths algorithms start with INIT-SINGLE-SOURCE.

INIT-SINGLE-SOURCE*(V, s)*

for each *v ∈*  *V*

*d*[*v*]←∞

*π*[*v*] ← NIL

*d*[*s*] ← 0

Relaxing an edge *(u, v) -* Can we improve the shortest-path estimate (best seen so far) from the source s to *v* by going through *u* and taking edge *(u, v)*?

|  |  |
| --- | --- |
| RELAX*(u, v, w)*  if *d*[*v*] *> d*[*u*] + *w(u, v)*  *d*[*v*] ← *d*[*u*] + *w(u, v)*  *π*[*v*]← *u* |  |

**The algorithms differ in the order and how many times they relax each edge.**

Shortest Path Algorithm - Bellman-Ford

The most straightforward of the “relax an edge” algorithms. Relaxes the edges in a fixed order (any fixed order) |v|-1 times. Not a greedy algorithm.

* Allows negative-weight edges.
* Computes *d*[*v*] and *π*[*v*] for all *v ∈*  *V*.
* Returns TRUE if no negative-weight cycles reachable from *s*, FALSE otherwise.

|  |  |
| --- | --- |
| BELLMAN-FORD*(V, E, w, s)*  INIT-SINGLE-SOURCE*(V, s)*  **for** *i* ← 1 to |*V*|-1  **for** each edge *(u, v) ∈*  *E* RELAX*(u,v,w)*  **for** each edge *(u, v) ∈*  *E* ***// all edges, in any order, same order each time*** **if** *d*[*v*] *> d*[*u*] + *w(u, v)*  **then return** FALSE  **return** TRUE |  |

3. Execute Bellman-Ford on the above graph from source s for this edge order (*t, x*), (*t, y*), (*t, z*), (*x, t*), (*y, x*), (*y, z*), (*z, x*), (*z, s*), (*s, t*), (*s, y*). Update the d[v] and π[v] values for each iteration.

4. What is the runtime of Bellman-Ford?

5. Prove Bellman-Ford is correct.

Values you get on each pass and how quickly it converges depends on order of relaxation. But guaranteed to converge after |V|-1 passes, assuming no negative-weight cycles.