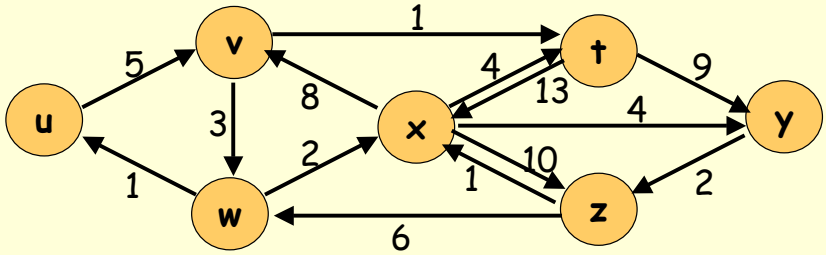
CSE 202 - Algorithms

Shortest Paths Problems

Shortest Paths Problems



Given a weighted directed graph,

- <u,v,t,x,z> is a <u>path</u> of weight 29 from u to z.
- <u,v,w,x,y,z> is another path from u to z; it has weight 16 and is the <u>shortest path</u> from u to z.

Variants of Shortest Paths Problems

A. Single pair shortest path problem

- Given s and d, find shortest path from s to d.

B. Single source shortest paths problem

- Given s, for each d find shortest path from s to d.

C. All-pairs shortest paths problem

- For each ordered pair s,d, find shortest path.

(1) and (2) seem to have same asymptotic complexity.

(3) takes longer, but not as long as repeating (2) for each s.

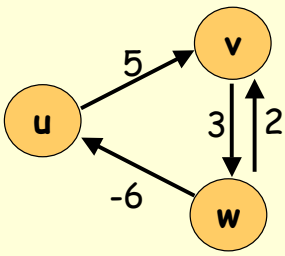
More Shortest Paths Variants

1. All weights are non-negative.
2. Weights may be negative, but no negative cycles

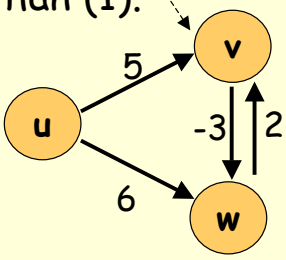
   (A cycle is a path from a vertex to itself.)
3. Negative cycles allowed.

   Algorithm reports "- ∞" if there is a negative cycle on path from source to destination

(2) and (3) seem to be harder than (1).



	u	v	w
u	0	5	8
v	-3	0	3
w	-6	-1	0



	u	v	w
u	0	-∞	-∞
v	∞	-∞	-∞
w	∞	-∞	-∞

Single Source Shortest Paths

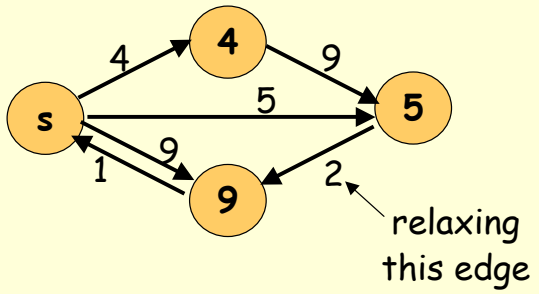
Non-negative weights (problem B-1):

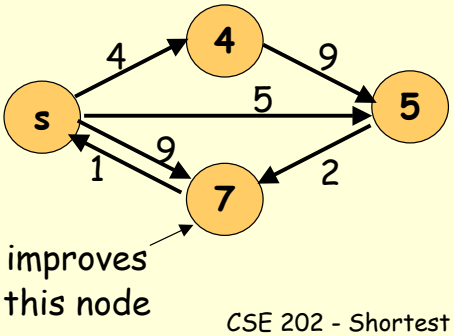
From source, construct *tree* of shortest paths.

- Why is this a tree?
- Is this a Minimum Spanning Tree?

Basic approach: label nodes with shortest path found so far.

Relaxation step: choose any edge (u,v) . If it gives a shorter path to v, update v 's label.





Single Source Shortest Paths

Simplest strategy:

- Keep cycling through all edges
- When all edges are relaxed, you're done.

  What is upper bound on work?

Improvement:

We mark nodes when we're sure we have best path.

Key insight: if you relax all edges out of marked nodes, the unmarked node that is closest to source can be marked.

This gives Dijkstra's algorithm

Relax edge (u,v) only once - just after u is marked.

Keep nodes on min-priority queue – need E Decrease-Key's.

Gives O(E lg V) (or O(E + V lg V) using Fibonacci heap.)

There's a simple $O(V^2)$ implementation – when is it fastest?

Single Source Shortest Paths

Negative weights allowed (problem B-2 and B-3)

Why can't we just add a constant to each weight?

Why doesn't Dijkstra's algorithm work for B-2 ?

Bellman-Ford: Cycle through edges V-1 times.

O(VE) time.

PERT chart: Arbitrary weights, graph is acyclic:

Is there a better algorithm than Bellman-Ford?

All-pairs Shortest Paths

Naïve #1: Solve V single-source problems.

- $O(V^2 E)$ for general problem.
- $O(V^3)$ or $O(V E \lg V)$ for non-negative weights.

Naïve #2: Let $d_k(i,j)$ = shortest path from i to j involving $\leq$ k edges.

$d_1(i,j)$ = is original weight matrix.

Compute d_{k+1}'s from d_k's by seeing if adding edge helps:

$$d_{k+1}(i,j) = \text{Min} \{ d_k(i,m) + d_1(m,j) \}$$

Hmmm ...this looks a lot like matrix multiplication

If there are no negative cycles, $d_{V-1}(i,j)$ is solution.

If there is a negative cycle, then $d_{V-1} \neq d_V$

Complexity is $O(V^4)$

All-pairs Shortest Paths

No negative cycles – Divide and Conquer says:

"Shortest path from a to b with at most 2^{k+1} hops is a shortest path from a to c with at most 2^k hops followed by one from c to b with at most 2^k hops."

$T(k+1) = T(k) + V^3$ so $T(\lg V)$ is $O(V^3 \lg V)$.

This also looks like matrix multiplication, using

$d_{2k} = d_k \times d_k$ instead of $d_{k+1} = d_k \times d_1$

All-pairs Shortest Paths

No negative cycles – Dynamic Programming says:

"Number cities 1 to V. Shortest path from a to b that only uses first cities 1 to k+1 as intermediate points is a path that goes from a to k+1 using only cities 1 to k, followed by a path going from k+1 to b (or shortest from a to b avoiding k+1 altogether)."

$T(k+1) = T(k) + cV^2$, and $T(0)$ is 0.

Thus, $T(V)$ is $O(V^3)$.

This is the <u>Floyd-Warshall</u> algorithm

Johnson's All-Pairs Shortest Paths

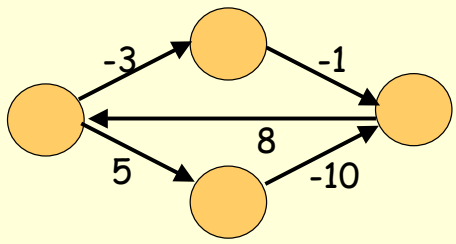
Motivation: for sparse non-negative weights,

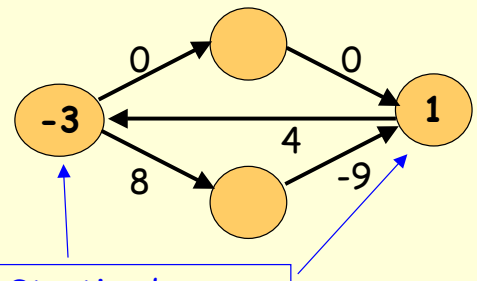
"O(V E lg V)" ← Dijkstra V times

is better than

"$O(V^3)$" ← Floyd-Warshall

We'll convert "arbitrary weights" (C3) to "non-negative" (C1) problem via <u>reweighting</u>.

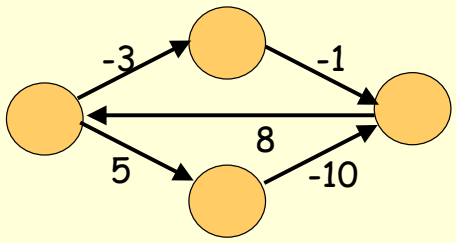


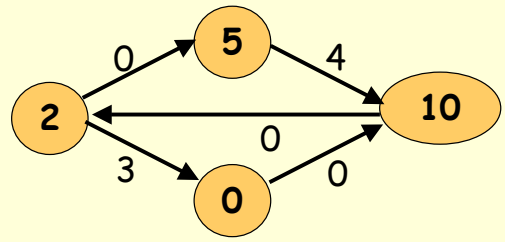


Starting bonus
or finishing penalty

Johnson's Algorithm

Reweighting can make all edges non-negative





Make node-weight(x) = shortest path to x from anywhere.

Sounds like all-pairs problem

Slick trick
use Bellman-Ford
(only O(VE) time)