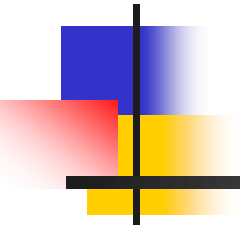
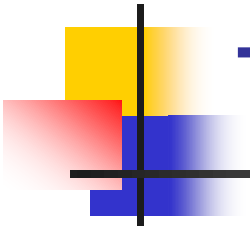


Algorithms and Data Structures
Lecture XIII



This Lecture

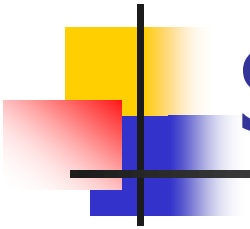
- Single-source shortest paths in weighted graphs
  - Shortest-Path Problems
  - Properties of Shortest Paths, Relaxation
  - Dijkstra's Algorithm
  - Bellman-Ford Algorithm
  - Shortest-Paths in DAG's

Shortest Path

- Generalize distance to weighted setting
- Digraph $G = (V,E)$ with weight function W: $E \rightarrow R$ (assigning real values to edges)
- Weight of path $p = v_1 \rightarrow v_2 \rightarrow \ldots \rightarrow v_k$ is

$$w(p) = \sum_{i=1}^{k-1} w(v_i, v_{i+1})$$

- Shortest path = a path of the minimum weight
- Applications
  - static/dynamic network routing
  - robot motion planning
  - map/route generation in traffic



Shortest-Path Problems

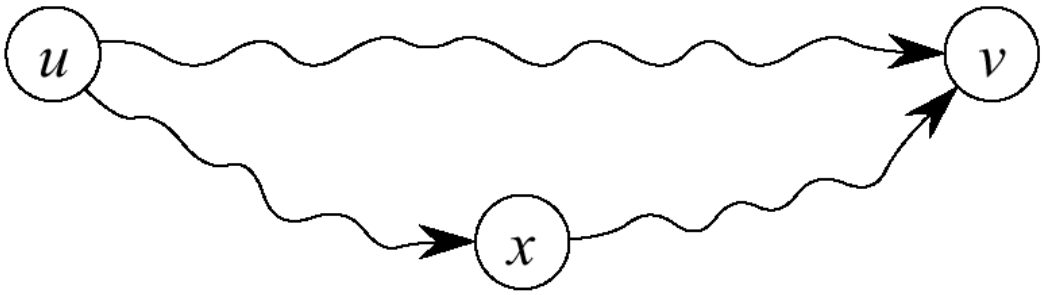
- Shortest-Path problems
  - **Single-source (single-destination).** Find a shortest path from a given source to each of the vertices
  - **Single-pair.** Given two vertices, find a shortest path between them. Solution to single-source problem solves this problem efficiently, too.
  - **All-pairs.** Find shortest-paths for every pair of vertices. Dynamic programming algorithm.
  - Unweighted shortest-paths – BFS.

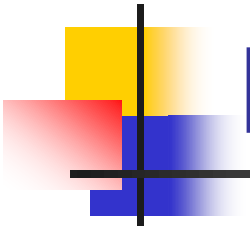
Optimal Substructure

- Theorem: subpaths of shortest paths are shortest paths
- Proof (cut and paste)
  - if some subpath were not the shortest path, one could substitute the shorter subpath and create a shorter total path

Triangle Inequality

- Definition
  - $\delta(u,v) \equiv$ weight of a shortest path from u to v
- Theorem
  - $\delta(u,v) \leq \delta(u,x) + \delta(x,v)$ for any x
- Proof
  - shortest path u Î v is no longer than any other path u Î v – in particular, the path concatenating the shortest path u Î x with the shortest path x Î v





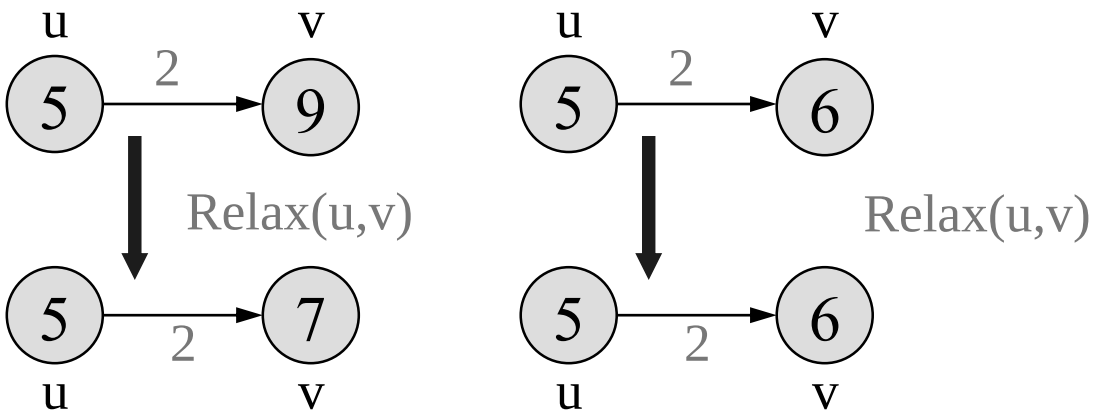
Negative Weights and Cycles?

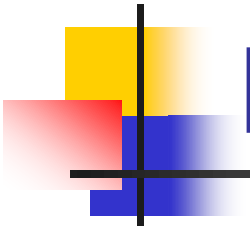
- Negative edges are OK, as long as there are no *negative weight cycles* (otherwise paths with arbitrary small "lengths" would be possible)
- Shortest-paths can have no cycles (otherwise we could improve them by removing cycles)
  - Any shortest-path in graph G can be no longer than $n - 1$ edges, where n is the number of vertices

Relaxation

- For each vertex in the graph, we maintain d[v], the shortest path estimate, initialized to ∞ at start
- Relaxing an edge (u,v) means testing whether we can improve the shortest path to v found so far by going through u

```
Relax (u,v,w)
if d[v]>d[u]+w(u,v) then
    d[v] ← d[u]+w(u,v)
    π[v] ← u
```





Dijkstra's Algorithm

- Non-negative edge weights
- Greedy, similar to Prim's algorithm for MST
- Like breadth-first search (if all weights = 1, one can simply use BFS)
- Use Q, priority queue keyed by $d[v]$ (BFS used FIFO queue, here we use a PQ, which is re-ordered whenever d decreases)
- Basic idea
  - maintain a set S of solved vertices
  - at each step select "closest" vertex u, add it to S, and relax all edges from u

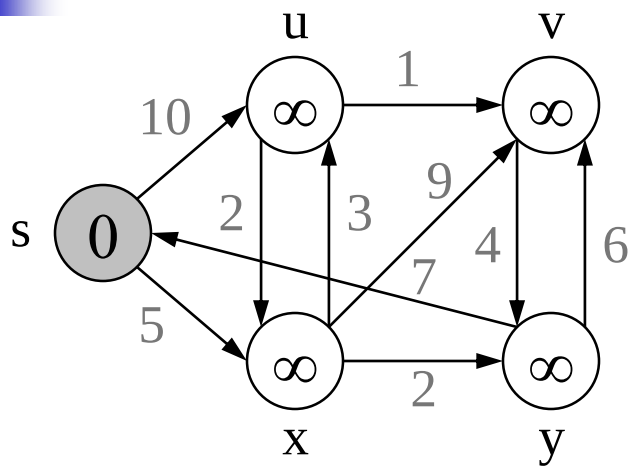
Dijkstra's Pseudo Code

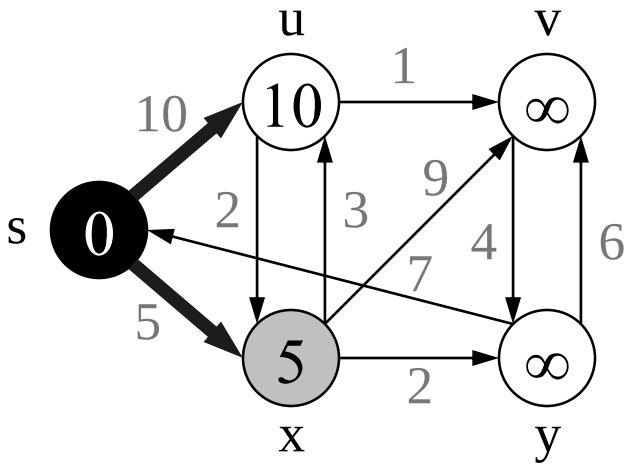
- Graph *G*, weight function *w*, root *s*

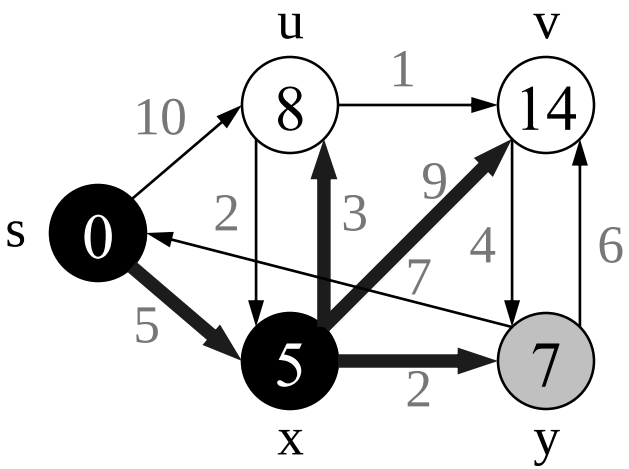
```
DIJKSTRA(G, w, s)
 1 for each v ∈ V
 2       do d[v] ← ∞
 3 d[s] ← 0
 4 S ← ∅   ▷ Set of discovered nodes
 5 Q ← V
 6 while Q ≠ ∅
 7       do u ← EXTRACT-MIN(Q)
 8          S ← S ∪ {u}
 9          for each v ∈ Adj[u]
10              do if d[v] > d[u] + w(u, v)
11                    then d[v] ← d[u] + w(u, v)
```

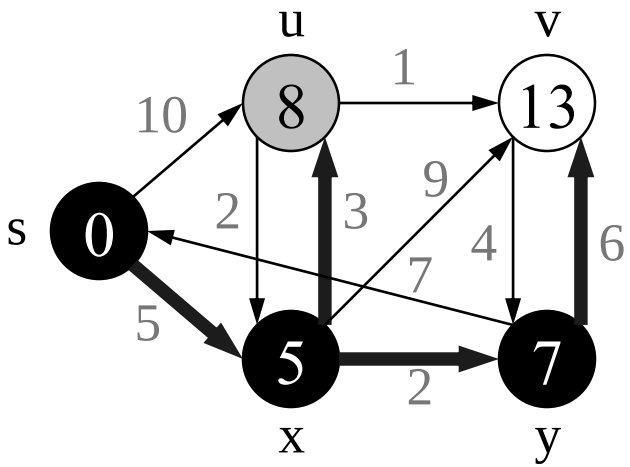
relaxing edges

Dijkstra's Example

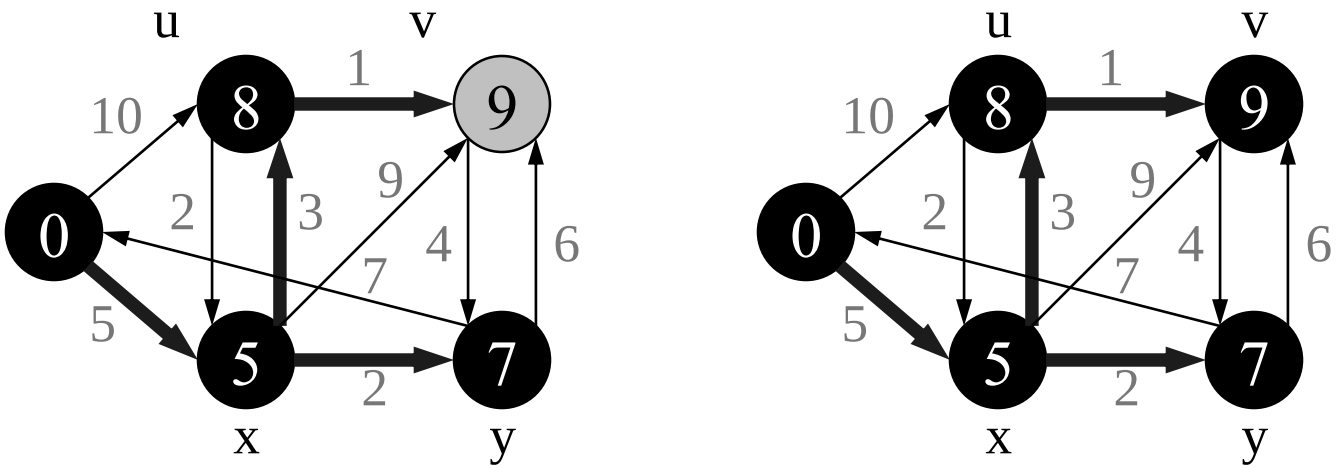








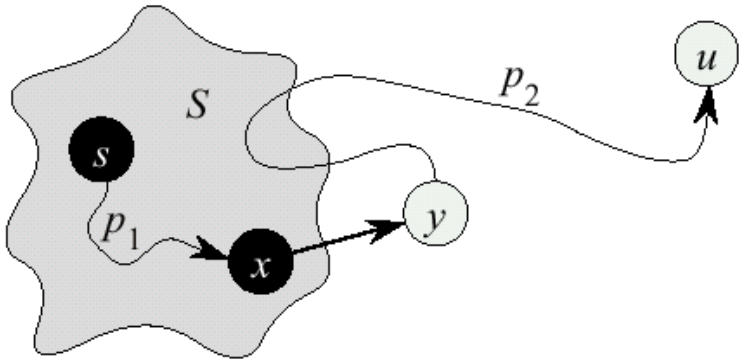
Dijkstra's Example (2)



- Observe
  - relaxation step (lines 10-11)
  - setting $d[v]$ updates Q (needs Decrease-Key)
  - similar to Prim's MST algorithm

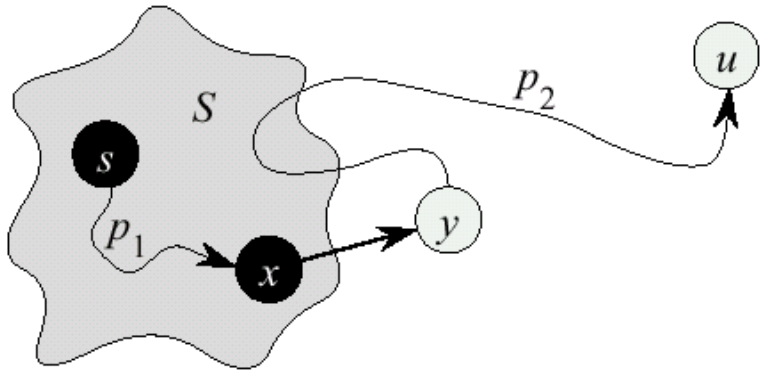
Dijkstra's Correctness

- We will prove that **whenever u is added to S**, $d[u] = \delta(s,u)$, i.e., that d is minimum, and that equality is maintained thereafter
- Proof
  - Note that $\forall v,\ d[v] \geq \delta(s,v)$
  - Let u be the first **vertex picked** such that $\exists$ shorter path than $d[u]$, i.e., that $\Rightarrow d[u] > \delta(s,u)$
  - We will show that the assumption of such a vertex leads to a contradiction



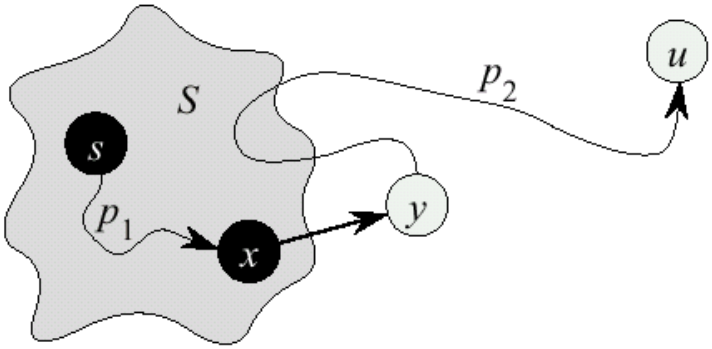
Dijkstra Correctness (2)

- Let y be the first vertex $\in V - S$ on the actual shortest path from s to u, then it must be that $d[y] = \delta(s,y)$ because
  - $d[x]$ is set correctly for y's predecessor $x \in S$ on the shortest path (by choice of u as the first vertex for which d is set incorrectly)
  - when the algorithm inserted x into S, it relaxed the edge (x,y), assigning $d[y]$ the correct value



Dijkstra Correctness (3)

$$
\begin{aligned}
d[u] &> \delta(s,u) && \text{(initial assumption)} \\
&= \delta(s,y) + \delta(y,u) && \text{(optimal substructure)} \\
&= d[y] + \delta(y,u) && \text{(correctness of } d[y]\text{)} \\
&\geq d[y] && \text{(no negative weights)}
\end{aligned}
$$

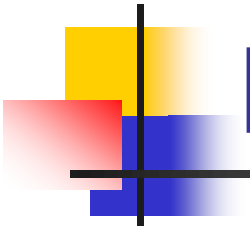


- But $d[u] > d[y] \Rightarrow$ algorithm would have chosen y (from the PQ) to process next, not $u \Rightarrow$ Contradiction
- Thus d[u] = $\delta(s,u)$ at time of insertion of u into S, and Dijkstra's algorithm is correct

Dijkstra's Running Time

- Extract-Min executed $|V|$ time
- Decrease-Key executed $|E|$ time
- Time = $|V| \ T_{\text{Extract-Min}} + |E| \ T_{\text{Decrease-Key}}$
- T depends on different Q implementations

Q	T(Extract-Min)	T(Decrease-Key)	
array	$O(V)$	$O(1)$	$O(V^2)$
binary heap	$O(\lg V)$	$O(\lg V)$	$O(E \lg V)$
Fibonacci heap	$O(\lg V)$	$O(1)$	$O(V \lg V + E)$



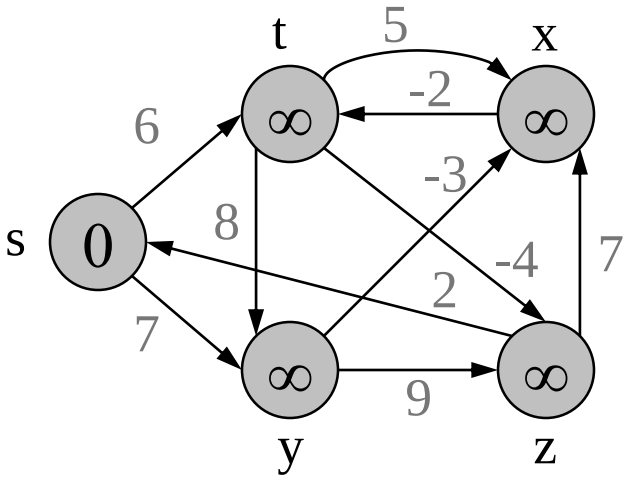
Bellman-Ford Algorithm

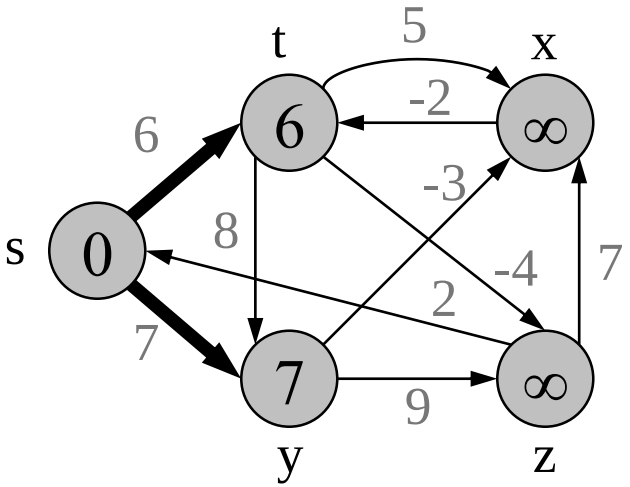
- Dijkstra's doesn't work when there are negative edges:
  - Intuition – we can not be greedy on the assumption that the lengths of paths will only increase in the future
- Bellman-Ford algorithm detects negative cycles (returns *false*) or returns the shortest path-tree

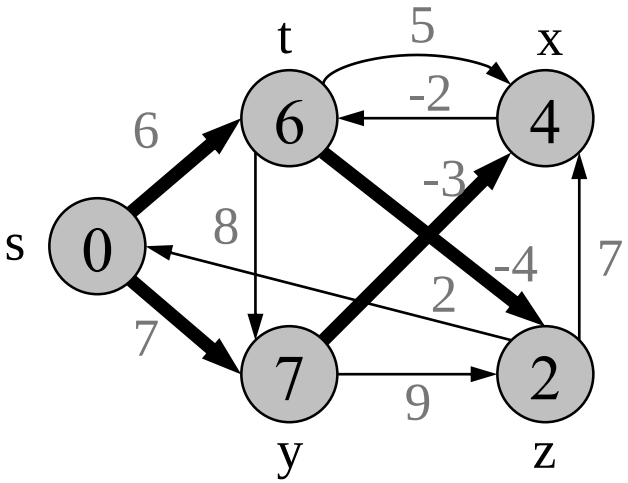
Bellman-Ford Algorithm

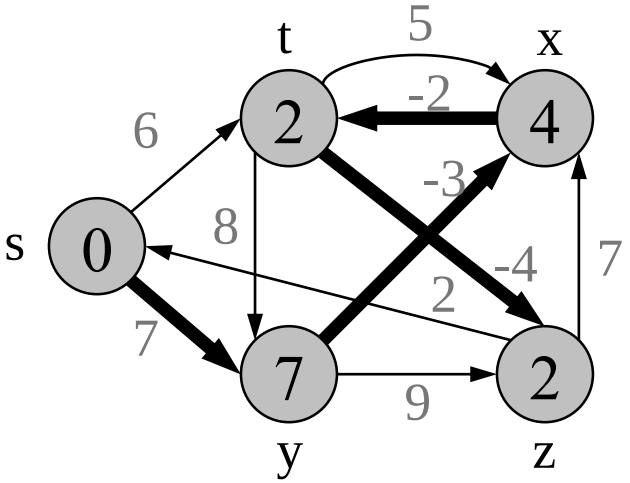
```
Bellman-Ford(G,w,s)
01 for each v ∈ V[G]
02     d[v] ← ∞
03 d[s] ← 0
04 π[r] ← NIL
05 for i ← 1 to |V[G]|-1 do
06     for each edge (u,v) ∈ E[G] do
07         Relax (u,v,w)
08 for each edge (u,v) ∈ E[G] do
09     if d[v] > d[u] + w(u,v) then return false
10 return true
```

Bellman-Ford Example

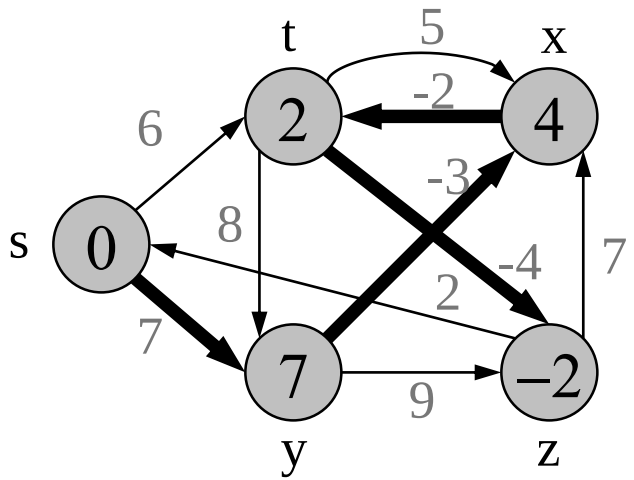








Bellman-Ford Example



- Bellman-Ford running time:
  - $(|V|-1)|E| + |E| = \Theta(VE)$

Correctness of Bellman-Ford

- Let $\delta_i(s,u)$ denote the length of path from s to u, that is shortest among all paths, that contain at most i edges
- Prove by induction that $d[u] = \delta_i(s,u)$ after the i-th iteration of Bellman-Ford
  - Base case, trivial
  - Inductive step (say $d[u] = \delta_{i-1}(s,u)$):
    - Either $\delta_i(s,u) = \delta_{i-1}(s,u)$
    - Or $\delta_i(s,u) = \delta_{i-1}(s,z) + w(z,u)$
    - In an iteration we try to relax each edge ((z,u) also), so we will catch both cases, thus $d[u] = \delta_i(s,u)$

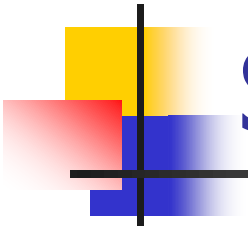
Correctness of Bellman-Ford

- After *n-1* iterations, $d[u] = \delta_{n-1}(s,u)$, for each vertex *u.*
- If there is still some edge to relax in the graph, then there is a vertex *u*, such that $\delta_n(s,u) < \delta_{n-1}(s,u)$. But there are only *n* vertices in *G* – we have a cycle, and it must be negative.
- Otherwise, $d[u] = \delta_{n-1}(s,u) = \delta(s,u)$, for all *u,* since any shortest path will have at most *n-1* edges

Shortest-Path in DAG's

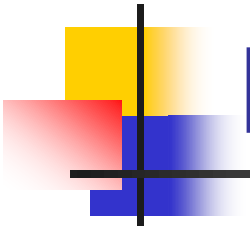
- Finding shortest paths in DAG's is much easier, because it is easy to find an order in which to do relaxations – Topological sorting!

```
DAG-Shortest-Paths(G,w,s)
01 for each v ∈ V[G]
02     d[v] ← ∞
03 d[s] ← 0
04 topologically sort V[G]
05 for each vertex u, taken in topolog. order do
06     for each vertex v ∈ Adj[u] do
07         Relax(u,v,w)
```



Shortest-Paths in DAG's (2)

- Running time:
  - $\Theta(V+E)$ – only one relaxation for each edge, V times faster than Bellman-Ford



Next lecture

- Introduction to Computational Geometry