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

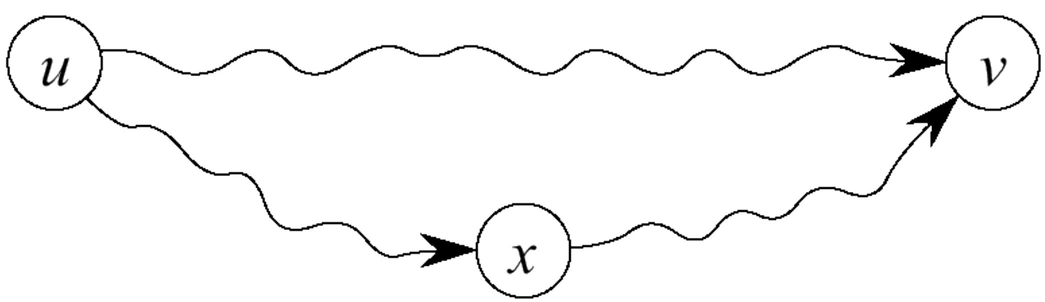
- Shortest-Path problems
  - **Single-source (single-destination).** Find a shortest path from a given source (vertex *s*) to each of the vertices. The topic of this lecture.
  - **Single-pair.** Given two vertices, find a shortest path between them. Solution to single-source problem solves this problem 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

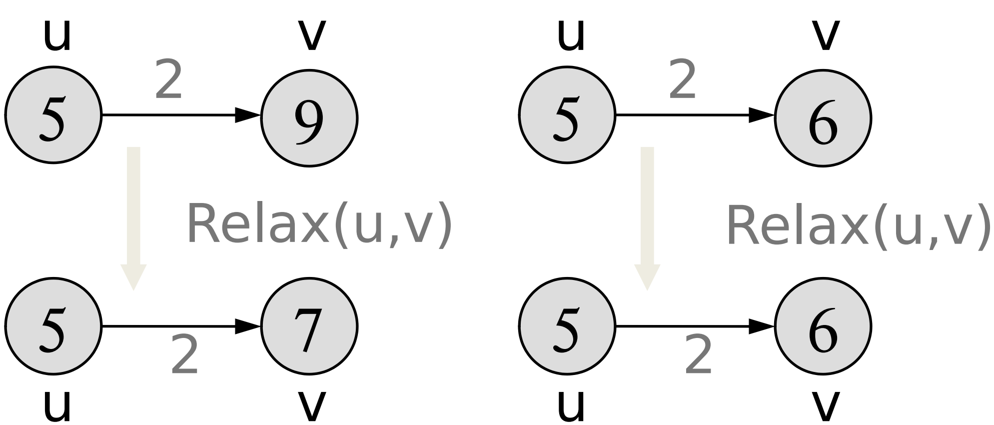


Negative Weights and Cycles?

- 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 estimate of the shortest path from *s*, 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-organized whenever some 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

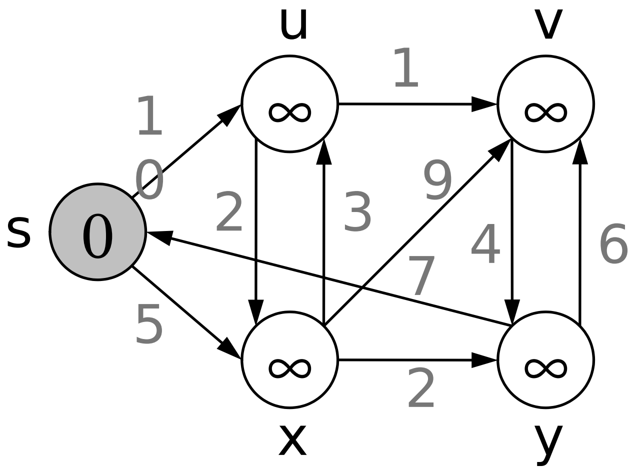
Dijkstra's Pseudo Code

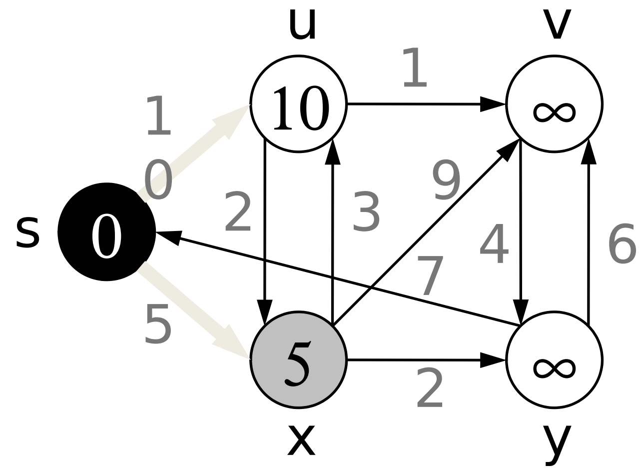
- Graph *G*, weight function *w*, root *s*

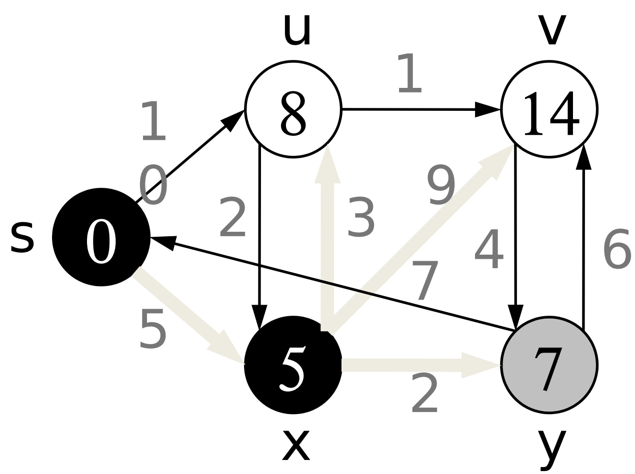
```
DIJKSTRA(G, w, s)
for each v ∈ V
        do d[v] ← ∞
d[s] ← 0
S ← ∅    ▷ Set of discovered nodes
Q ← V
while Q ≠ ∅
        do u ← EXTRACT-MIN(Q)
           S ← S ∪ {u}
           for each v ∈ Adj[u]
                do if d[v] > d[u] + w(u, v)
                        then d[v] ← d[u] + w(u, v)
```

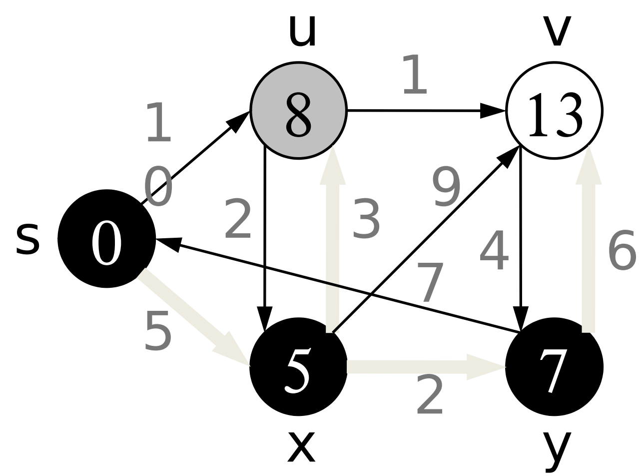
relaxin
g
edges

Dijkstra’s Example

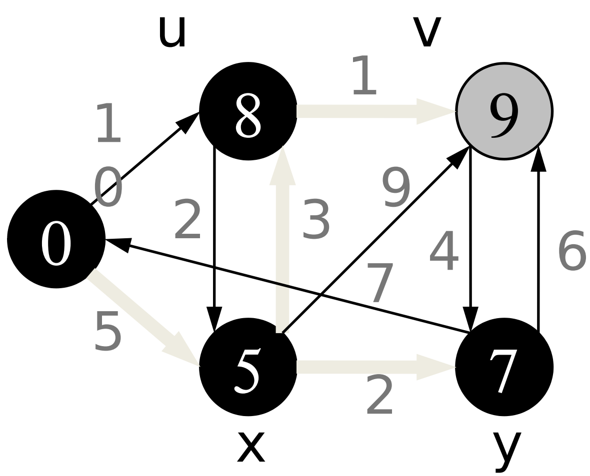


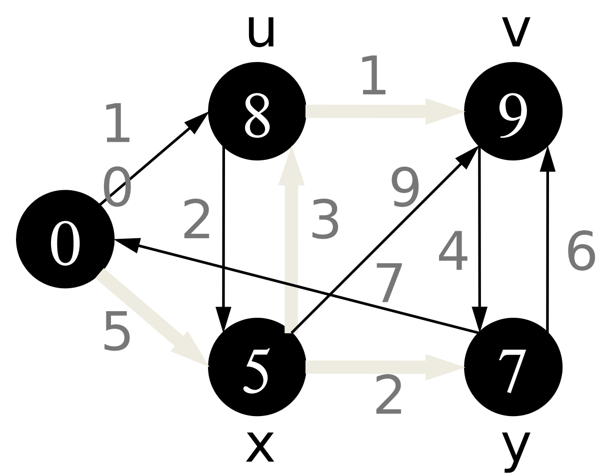






Dijkstra’s Example (2)

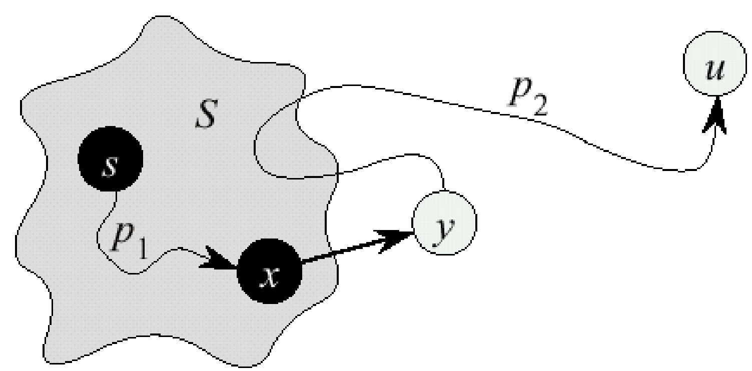




- Observe
  - relaxation step (lines 10-11)
  - setting *d*[*v*] updates *Q* (needs Decrease-Key)
  - similar to Prim's MST algorithm

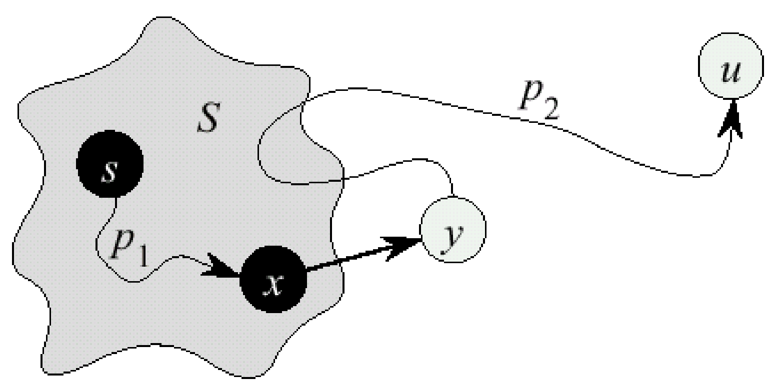
Dijkstra's Correctness

- 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 there is a shorter path than $d[u]$, i.e., that $\Rightarrow d[u] > \delta(s,u)$
  - We will show that this assumption leads to a contradiction



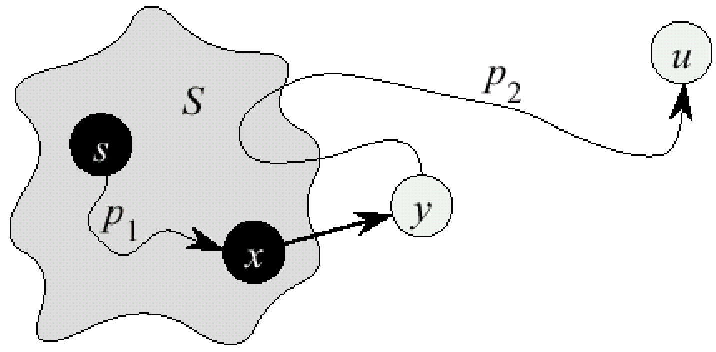
Dijkstra Correctness (2)

- 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)	Total
array	$O(V)$	$O(1)$	$O(V^2)$
binary heap	$O(\lg V)$	$O(\lg V)$	$O(E \lg V)$
Fibonacci heap	$O(\lg V)$	$O(1)$ (amort.)	$O(V \lg V + E)$