Algorithms

S-S Shortest Path: *Dijkstra's Algorithm*

Disjoint-Set Union

Amortized Analysis

Review: Single-Source Shortest Path

- Problem: given a weighted directed graph G, find the minimum-weight path from a given source vertex s to another vertex v
  - "Shortest-path" = minimum weight
  - Weight of path is sum of edges
  - E.g., a road map: what is the shortest path from Chapel Hill to Charlottesville?

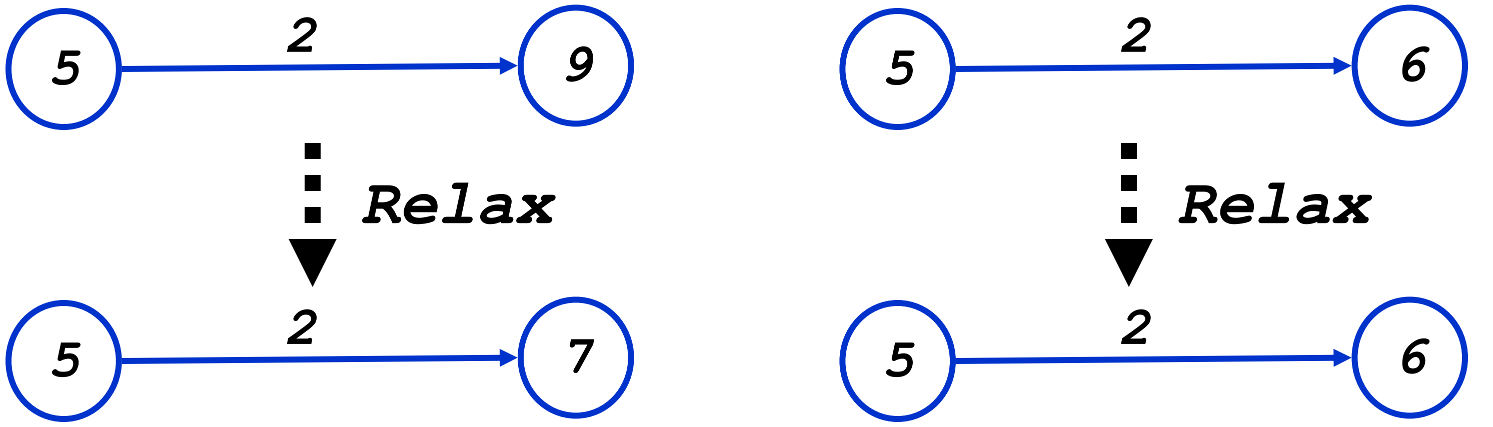
Review: Shortest Path Properties

- *Optimal substructure*: the shortest path consists of shortest subpaths
- Let $\delta(u,v)$ be the weight of the shortest path from u to v. Shortest paths satisfy the *triangle inequality*: $\delta(u,v) \leq \delta(u,x) + \delta(x,v)$
- In graphs with negative weight cycles, some shortest paths will not exist

Review: Relaxation

- Key technique: *relaxation*
  - Maintain upper bound d[*v*] on δ(s,v):

```
Relax(u,v,w) {
   if (d[v] > d[u]+w) then d[v]=d[u]+w;
}
```



Review: Bellman-Ford Algorithm

```
BellmanFord()
    for each v ∈ V
        d[v] = ∞;
    d[s] = 0;
    for i=1 to |V|-1
        for each edge (u,v) ∈ E
            Relax(u,v, w(u,v));
    for each edge (u,v) ∈ E
        if (d[v] > d[u] + w(u,v))
            return “no solution”;
```

Initialize d[], which will converge to shortest-path value δ

Relaxation: Make |V|-1 passes, relaxing each edge

Test for solution: have we converged yet? Ie, ∃ negative cycle?

```
Relax(u,v,w): if (d[v] > d[u]+w) then d[v]=d[u]+w
```

Review: Bellman-Ford Algorithm

```
BellmanFord()
   for each v ∈ V
      d[v] = ∞;
   d[s] = 0;
   for i=1 to |V|-1
      for each edge (u,v) ∈ E
         Relax(u,v, w(u,v));
   for each edge (u,v) ∈ E
      if (d[v] > d[u] + w(u,v))
           return "no solution";
```

What will be the running time?

```
Relax(u,v,w): if (d[v] > d[u]+w) then d[v]=d[u]+w
```

Review: Bellman-Ford

- Running time: O(VE)
  - Not so good for large dense graphs
  - But a very practical algorithm in many ways
- Note that order in which edges are processed affects how quickly it converges (show example)

DAG Shortest Paths

- Problem: finding shortest paths in DAG
  - Bellman-Ford takes O(VE) time.
  - *How can we do better?*
  - Idea: use topological sort. *How does it work again?*
    - If were lucky and processes vertices on each shortest path from left to right, would be done in one pass
    - Every path in a dag is subsequence of topologically sorted vertex order, so processing verts in that order, we will do each path in forward order (will never relax edges out of vert before doing all edges into vert).
    - Thus: just one pass. *What will be the running time?*

Dijkstra's Algorithm

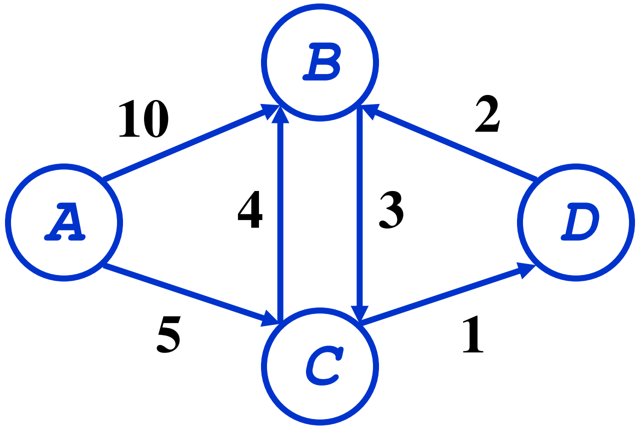
- If no negative edge weights, we can beat BF
- Similar to breadth-first search
    - Grow a tree gradually, advancing from vertices taken from a queue
- Also similar to Prim's algorithm for MST
    - Use a priority queue keyed on d[v]

Dijkstra's Algorithm

```
Dijkstra(G)
    for each v ∈ V
        d[v] = ∞;
    d[s] = 0; S = ∅; Q = V;
    while (Q ≠ ∅)
        u = ExtractMin(Q);
        S = S U {u};
        for each v ∈ u->Adj[]
            if (d[v] > d[u]+w(u,v))
                d[v] = d[u]+w(u,v);
```

$\}$ *Relaxation Step*

Note: this is really a call to Q->DecreaseKey()


$$\begin{tikzcd}
 & B \arrow[dd, shift left, "3"] & \\
A \arrow[ur, "10"] \arrow[dr, "5"'] & & D \arrow[ul, "2"'] \\
 & C \arrow[uu, shift left, "4"] \arrow[ur, "1"'] &
\end{tikzcd}$$

Ex: run the algorithm