More on Graphs
Minimum Spanning Trees

• Given a connected graph $G = (V, E)$ let $w[u, v]$ specify the “weight” of an edge $(u, v) \in E$

• A spanning tree for $G$ is a subset $T \subseteq E$ such that the graph $(V, T)$ is connected and acyclic
  ▪ In other words, $T$ is a tree containing all vertices in $V$

• The weight of spanning tree $T$ is given by:
  \[ W(T) = \sum_{(u, v) \in T} w[u, v] \]

• A Minimum Spanning Tree (MST) is a spanning tree of minimal weight
Growing an MST

• Start with an empty set of edges $A$, and repeatedly add “safe” edges, until $A$ is a spanning tree
  ▪ An edge $(u,v)$ is safe to add if it maintains the invariant that the set of edges $A$ is a subset of some MST for $G$

• Theorem 23.1 in CLRS:
  Let $A$ be a subset of some MST for $G$, and let $(S, V-S)$ be any cut of $G$ that respects $A$. If $(u,v)$ is a light edge crossing the cut then $(u,v)$ is safe for $A$.
  ▪ Proof?

• How can we find such safe edges efficiently?
Kruskal’s Algorithm

MST-Kruskal(V,E,w)
A ← ∅
for all v ∈ V
    Make-Set(v) // Each node initially in its own set
Sort E in non-decreasing order by weight w
for each (u,v) ∈ E // processed in sorted order
    if Find-Set(u) ≠ Find-Set(v) // if u,v in different sets
        A ← A ∪ {(u,v)} // (u,v) is a safe edge, add it
        Union(u,v) // then merge the sets for u and v
return A

• Runtime depends on Make-Set/Find-Set/Union ops
  ▪ More on those later…what about correctness?
Prim’s Algorithm

MST-Prim(V,E,w,r)  // r is to be the “root” of the MST
for all v ∈ V
key[v] = ∞,  π[v] = NIL
key[r] = 0
Q ← V  // Q is a min-priority queue (priorities in key[V])
while Q ≠ ∅
  u = Extract-Min(Q)  // u already connected “safely”
  for all v ∈ Adj[u]
    if v ∈ Q and w[u,v] < key[v]
return A = { (v, π[v]) : v ∈ V - {r} } 

• Runtime depends on the choice of Min-Priority-Queue
  ▪ Can get O( |V| 1g |V| ) using binary heaps

Shabsi Walfish  NYU - Fundamental Algorithms
Summer 2005
The Single-Source Shortest Path (SSSP) Problem

• A path $p$ is a sequence of vertices $(v_0, v_1, \ldots, v_k)$ such that $(v_{i-1}, v_i) \in E$ for $i=1 \ldots k$
  ▪ Notation: $u \leadsto v$ means a path from $u$ to $v$

• The weight of a path $p$ is $w(p) = \sum_{i=1}^{k} w(v_{i-1}, v_i)$

• The shortest-path weight from $u$ to $v$ is $\delta(u, v) = \min(w(u \leadsto v))$

• A shortest path $p$ has weight $w(p) = \delta(u, v)$

• Single-Source Shortest Path Problem: Find the shortest paths from source vertex $s$ to all other vertices in the graph
Path Relaxation

Properties of Shortest Paths and Path Relaxation

Relax($u, v, w$) // Initially, for all $v \neq s$: $d[v] = \infty$, $\pi[v] = \text{NIL}$. $d[s] = 0$

- Triangle Inequality: $\delta(s, v) \leq \delta(s, u) + w[u, v]$

- Upper-bound Property: $d[v] \geq \delta(s, v)$

Convergence: if $s \to u \to v$ is a shortest path and $d[u] = \delta(s, u)$ invoking Relax($u, v, w$) yields $d[v] = \delta(s, v)$

Path Relaxation: if the edges of a shortest path from $s$ to $v$ are Relaxed in sequence, $d[v] = \delta(s, v)$
Bellman-Ford

Bellman-Ford(V,E,w,s)
  Initialize d[ ] and π[ ]
  for i = 1 to |V| - 1
    for each (u,v) ∈ E
      Relax(u,v,w)
    for each (u,v) ∈ E
      if d[v] > d[u] + w[u,v] return FALSE
  return TRUE

- Returns true for any graph with no negative-weight cycles
- Runtime is O( |V| |E| )
SSSP for DAGs

DAG-Shortest-Paths(V,E,w,s)
   topologically sort V   // Can only do this for a DAG
   Initialize d[ ] and π[ ]
   for each u ∈ V       // In topologically sorted order…
      forall v ∈ Adj[u]
         Relax(u,v,w)

• Correctness follows immediately from Path-Relaxation Property
• Runtime is Θ( |V|+|E| )
Dijkstra’s Algorithm

\[
\text{Dijkstra}(V,E,w,s) \quad // \text{Compare to MST-Prim…}
\]

\[
\text{forall } v \in V
\]
\[
\begin{align*}
\text{d}[v] &= \infty, \quad \pi[v] = \text{NIL} \\
\text{d}[s] &= 0, \quad S \leftarrow \emptyset
\end{align*}
\]

\[
\text{Q} \leftarrow V \quad // \text{Q is a min-priority queue (priorities in d[V])}
\]

\[
\text{while } Q \neq \emptyset \\
\begin{align*}
\text{u} &= \text{Extract-Min}(Q) \quad // \text{d}[u] \text{ already minimal} \\
S &\leftarrow S \cup \{u\}
\end{align*}
\]

\[
\begin{align*}
\text{forall } v \in \text{Adj}[u] \\
\text{Relax}(u,v,w) \quad // \text{Update priority of } v \text{ afterwards!}
\end{align*}
\]

- Works for graphs with no negative edge weights only
- Runtime same as MST-Prim