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?

Prim’s Algorithm

MST-Prim($V, E, w, r$) // $r$ is to be the “root” of the MST
for all $v \in V$
key[v] = $\infty$, $\pi[v] = \text{NIL}$
key[r] = 0
Q ← $V$ // $Q$ is a min-priority queue (priorities in key[V])
while $Q \neq \emptyset$
  $u = \text{Extract-Min}(Q)$ // $u$ already connected “safely”
  for all $v \in \text{Adj}[u]$
    if $v \in Q$ and $w[u, v] < \text{key}[v]$
      key[v] = $w[u, v]$, $\pi[v] = u$ // Update priority of $v$
return $A = \{ v, \pi[v] : v \in V - \{r\} \}$
- Runtime depends on the choice of Min-Priority-Queue
  - Can get $O(|V| \log |V|)$ using binary heaps

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: $u \rightsquigarrow v$ 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 $\delta(u, v) = \min(w(u \rightsquigarrow v))$
- 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
Properties of Shortest Paths and Path Relaxation

Relax(u,v,w) // Initialize, for all v ∈ S: d[v] = ∞, π[v] = NIL, d[S] = 0
if d[v] > d[u] + w(u,v)
d[v] = d[u] + w(u,v), π[v] = u

• Triangle Inequality: δ(s,v) ≤ δ(s,u) + w(u,v)
• Upper-bound Property: d[v] ≥ δ(s,v)
• Convergence: if s → u → v is a shortest path and d[u] = δ(s,u) invoking Relax(u,v,w) yields d[v] = δ(s,v)
• Path-Relaxation: if the edges of a shortest path from s to v are relaxed in sequence, d[v] = δ(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)
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 v ∈ V // In topologically sorted order...
    for all v ∈ Adj[u]
      Relax(u,v,w)
  • Correctness follows immediately from Path-Relaxation Property
  • Runtime is Θ(|V|+|E|)

Dijkstra’s Algorithm

Dijkstra(V,E,w,s) // Compare to MST-Prim...
for all v ∈ V
  d[s] = ∞, π[v] = NIL
  d[S] = 0, S ← ∅
  Q ← V // Q is a min-priority queue (priorities in d[V])
  while Q ≠ ∅
    u = Extract-Min(Q) // d[u] already minimal
    S ← S ∪ {u}
    for all v ∈ Adj[u]
      Relax(u,v,w) // Update priority of v afterwards!

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