ch23: Minimum Spanning Tree
ch24: Single-Source Shortest Path

12 Nov 2013
CMPT231
Dr. Sean Ho
Trinity Western University

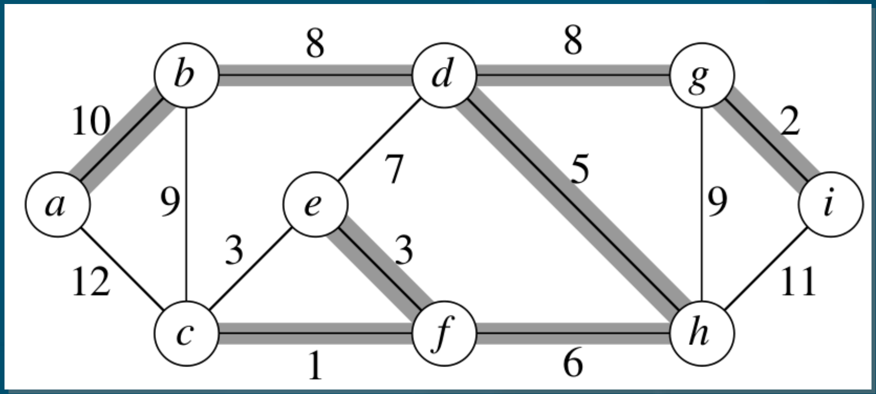
TRINITY WESTERN UNIVERSITY

Outline for today

- Minimum spanning tree
  - Generic solution outline
  - Kruskal's algorithm (builds a forest)
  - Prim's algorithm (builds a tree)
  - Uniqueness of MST
- Single-source shortest paths
  - Properties / lemmas about shortest paths
  - Bellman-Ford algorithm (neg weight allowed)
  - Special case if no cycles (DAG)
  - Dijkstra's algorithm (no neg weight)

Minimum spanning tree

- Given a connected, undirected graph G=(V,E)
  - with weights w(u,v) on each edge (u,v) ∈ E
- Find tree T ⊆ E that
  - Connects all vertices
  - Minimising total weight $w(T) = \Sigma_T\, w(u,v)$



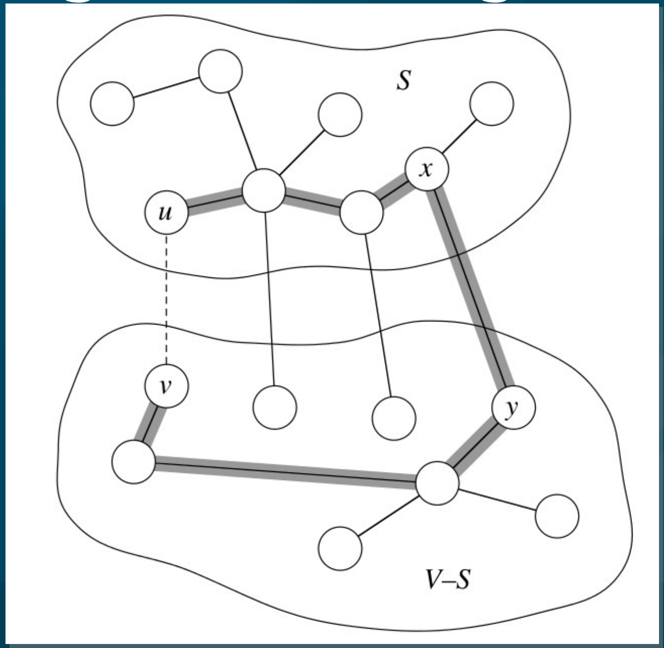
- Why must T be a tree? # edges in T? Unique?
- Applications: elec wiring in Moravia (Borůvka)
  - Utilities: gas/elec/water, Internet routing
  - Image analysis, registration, handwriting recog

Generic MST solution outline

- Build up solution A one edge at a time:
  - Start with $A = \emptyset$
  - while A is not a spanning tree:
    - find a safe edge (u,v) to add
    - add it to A
  - Loop iterates exactly $|V|-1$ times
- What do we mean by a safe edge (u,v)?
  - if A is a subset of a MST, then $A \cup (u,v)$ is still a subset of some MST
  - So adding (u,v) to A doesn't prevent us from finding a MST
- How do we find safe edges?

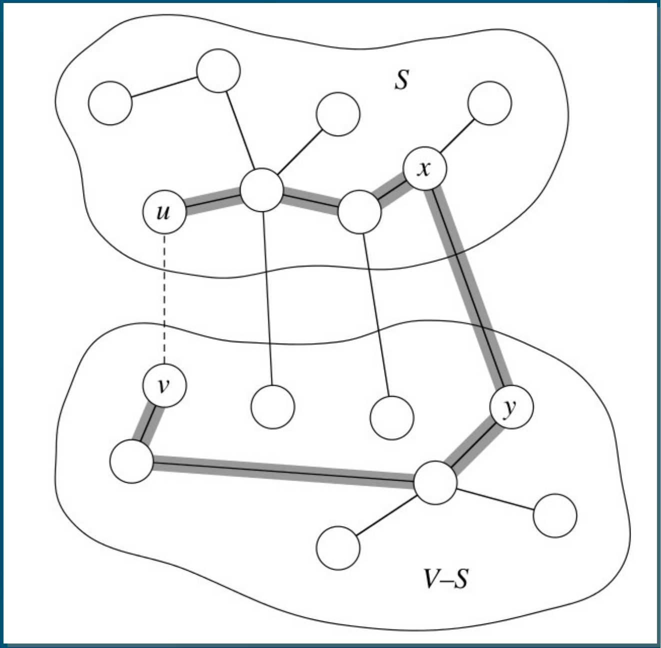
Safe edges across the cut

- Let A ⊆ E be a subset of some MST:
  - Let (S, V-S) be a cut: partitions the vertices
  - An edge (u,v) crosses the cut iff u ∈ S, v ∈ V-S
  - A cut respects A iff no edge in A crosses the cut
  - An edge is a light edge crossing the cut iff its weight is minimum over all edges crossing cut
- Theorem: Any light edge (u,v) crossing a cut (S, V-S) that respects A
  ⇒ (u,v) is a safe edge for A



solid lines: T
highlight: path u → v

Proof of safe edge theorem



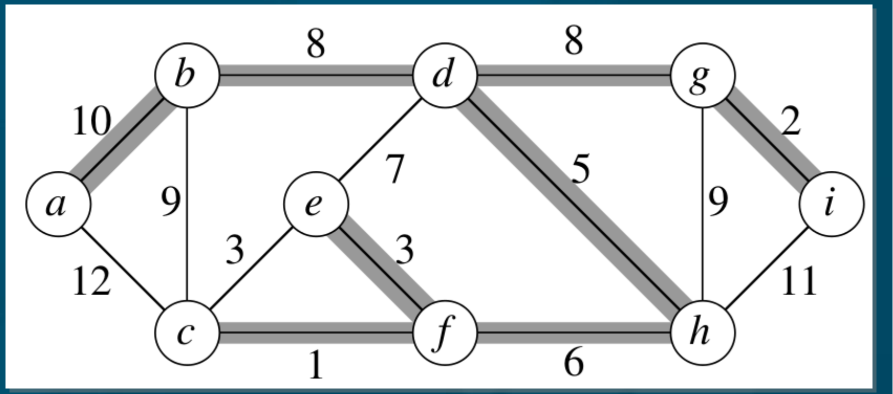
- Proof by "cut-and-paste":
  - Let T be a MST such that $A \subseteq T$
  - If $(u,v) \notin T$, modify T so it is
- T is a tree ⇒ ∃ unique path $u \to v$
- Path must cross the cut (S,V-S): pick a crossing edge, call it (x,y)
  - Since cut respects A, $(x,y) \notin A$
- (u,v) is a light edge crossing cut ⇒ $w(u,v) \le w(x,y)$
- Swap out edge: let $T' = T - \{(x,y)\} \cup \{(u,v)\}$:
  - $w(T') \le w(T)$, so T' is also a MST
  - $A \cup \{(u,v)\} \subseteq T'$, so (u,v) is safe for A

Outline for today

- Minimum spanning tree
  - Generic solution
  - Kruskal's algorithm
  - Prim's algorithm
  - Uniqueness of MST
- Single-source shortest paths
  - Properties / lemmas about shortest paths
  - Bellman-Ford algorithm
  - Special case if no cycles (DAG)
  - Dijkstra's algorithm

Kruskal's algorithm

- Each vertex starts as its own component
- Merge components by choosing light edges
  - Scan edge list in increasing order of weight
- Use disjoint-set ADT to find crossing edges
  - Kruskal(V, E, w):
    - A = ∅
    - for each v ∈ V: MakeSet(v)
    - sort E by weight w
    - for each (u,v) ∈ E in order:
      - if FindSet(u) ≠ FindSet(v): // crossing, so safe
        - A = A ∪ {(u,v)}
        - Union(u, v)
    - return A



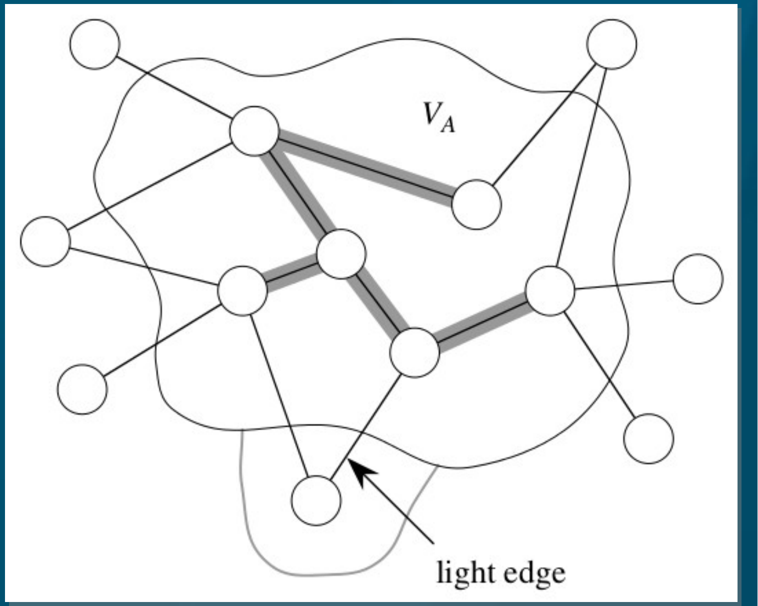
Kruskal: running time

- Init components: |V| * MakeSet
- Sort edge list by weight: |E| lg(|E|)
- Main for loop: |E| * FindSet, and |V| * Union
- Disjoint-set forest with union by rank and path compression (see ch21):
  - FindSet and Union are O(α(V))
    - α() is the inverse of the Ackermann function; very slow growing, $\alpha(n) \le 4$ for all reasonable n
  - $\Rightarrow O(V + E \lg E + E\,\alpha(V) + V\,\alpha(V)) = O(E \lg E)$
    - note that $|V| - 1 \le |E| \le |V|^2$
  - Even better if edges pre-sorted: O(E α(V)), essentially linear time in |E|

Outline for today

- Minimum spanning tree
  - Generic solution
  - Kruskal's algorithm
  - Prim's algorithm
  - Uniqueness of MST
- Single-source shortest paths
  - Properties / lemmas about shortest paths
  - Bellman-Ford algorithm
  - Special case if no cycles (DAG)
  - Dijkstra's algorithm

Prim's algorithm

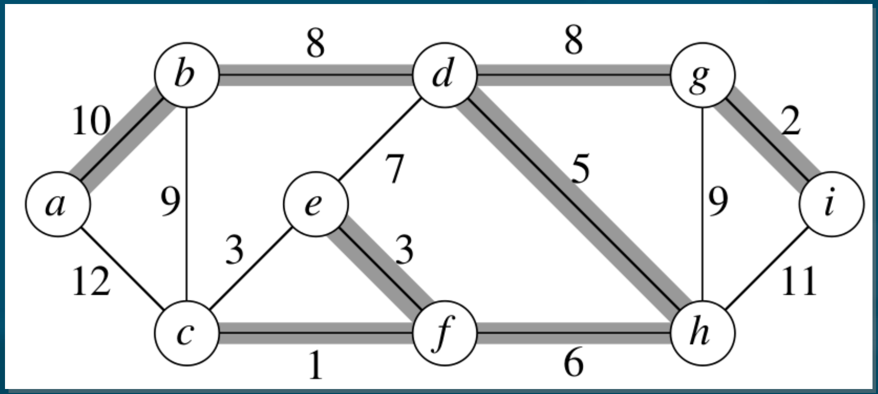


- Start from arbitrary root r
- Build tree by adding light edges crossing (V_A, $V-V_A$)
  - V_A = vertices incident on A
- Use priority queue Q to store vertices in $V-V_A$:
  - key (priority) of vertex v is min{$w(u,v)$: $u \in V_A$}
- So ExtractMin() returns v such that (u,v) is a light edge crossing (V_A, $V-V_A$)
- At each iteration, A is always a tree, and
  - A = {(v, v.parent): $v \in V$ – {r} – Q}
  - Final MST is encoded in the parent links

Prim: pseudocode

v	key	prt
a	∞	-
b	∞	-
c	∞	-
d	∞	-
e	∞	-
f	∞	-
g	∞	-
h	∞	-
i	∞	-

- Prim(V, E, w, r):
  - Q = new PriorityQueue(V)
  - DecreaseKey(Q, r, 0)   // set r.key = 0
  - while Q not empty:
    - u = ExtractMin(Q)
    - for each v ∈ Adj(u):
      - if v ∈ Q and w(u,v) < v.key:
        - v.parent = u
        - DecreaseKey(Q, v, w(u,v))



Prim: running time

- Initialise queue: |V| * Insert
- 1 * DecreaseKey on root
- Main loop: |V| * ExtractMin + O(E) * DecreaseKey
- Using binary max-heaps to implement queue:
  - Insert, ExtractMin, DecreaseKey are all O(lg V)
  - ⇒ O(V lg V + lg V + V lg V + E lg V)
  - = O(E lg V)
- Using Fibonacci heaps (ch19) instead, DecreaseKey can be done in O(1) amortised time
  - ⇒ O(V lg V + E)

Outline for today

- Minimum spanning tree
  - Generic solution
  - Kruskal's algorithm
  - Prim's algorithm
  - Uniqueness of MST
- Single-source shortest paths
  - Properties / lemmas about shortest paths
  - Bellman-Ford algorithm
  - Special case if no cycles (DAG)
  - Dijkstra's algorithm

Uniqueness of MST

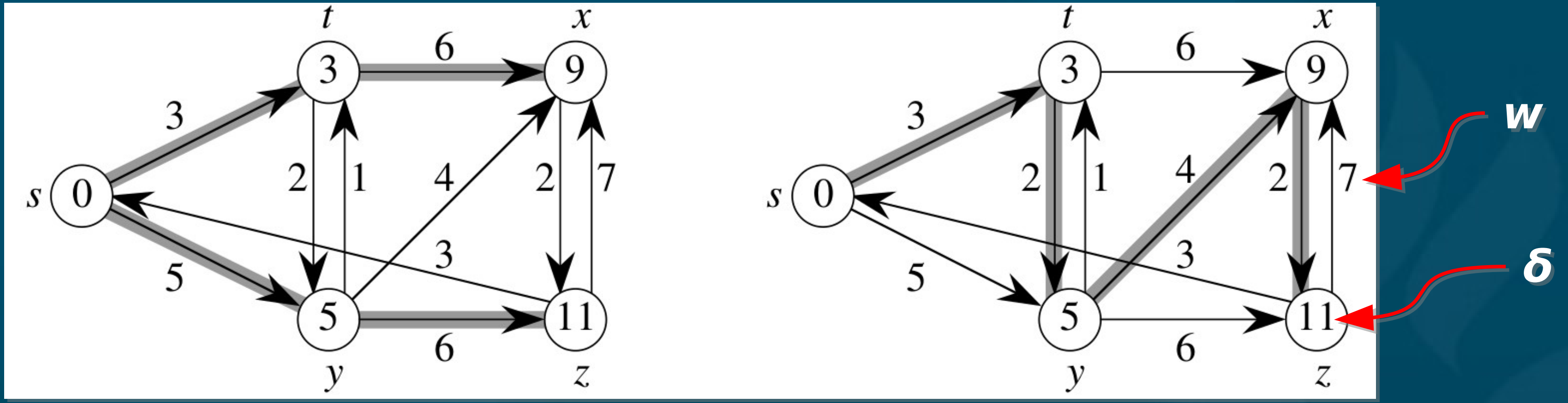
- In general, there may be multiple MSTs for a graph
- (p.630, 23.1-6): If every cut of the graph has a unique light edge crossing it, then MST is unique:
  - Let T, T' be two MSTs of the graph
  - Let (u,v) ∊ T. We want to show (u,v) ∊ T'
- T is a tree ⇒ T – {(u,v)} gives a cut: call it (S, V-S)
- (u,v) is a light edge crossing (S, V-S) (ex 23.1-3)
- T' must cross the cut, too: call its edge (x, y)
  - (x,y) is also a light edge crossing (S, V-S)
- By assumption, the light edge is unique
  - Hence (u,v) = (x,y), and so (u,v) ∊ T'

Outline for today

- Minimum spanning tree
  - Generic solution
  - Kruskal's algorithm
  - Prim's algorithm
  - Uniqueness of MST
- Single-source shortest paths
  - Properties / lemmas about shortest paths
  - Bellman-Ford algorithm
  - Special case if no cycles (DAG)
  - Dijkstra's algorithm

Shortest-path problems

- Input: directed graph G=(V, E), edge weights w
- Task: Find shortest paths between vertices
  - For a path p = (v_0, …, v_k): w(p) = Σ w(v_{i-1}, v_i)
  - Shortest-path weight δ(u,v) = min(w(p))
    - (or ∞ if v is not reachable from u)

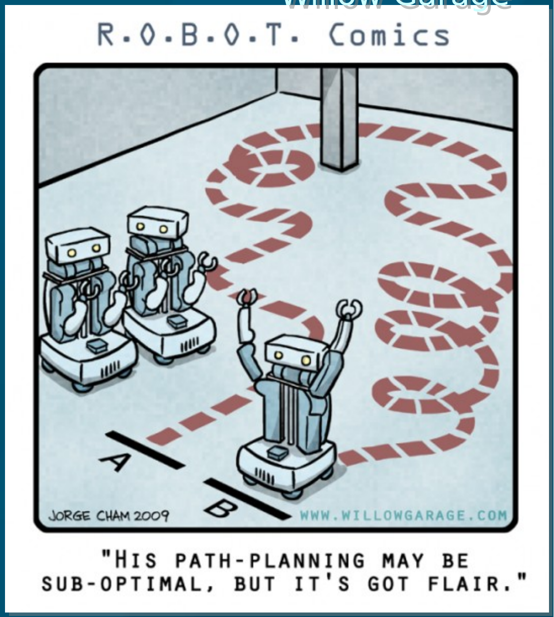


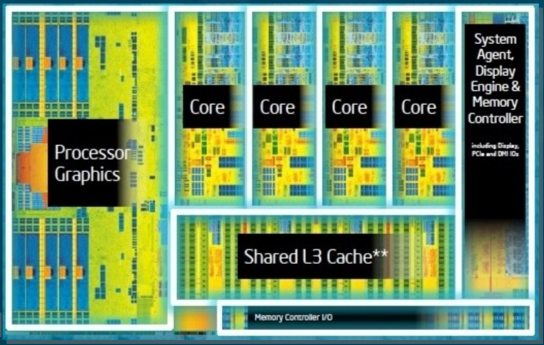
- Shortest-path not always unique
- Organised as tree rooted at source

Applications of shortest-path

- GPS/maps: turn-by-turn directions
  - All-pairs: optimise a fleet of trucks
  - Logistics / operations research
- Networking: optimal routing
- Robotics, self-driving: path planning
- Layout: factory/plant, VLSI chip design
- “Six degrees”: path to a celebrity
- Solving puzzles a la Rubik's Cube:
  - V = states, E = transitions/moves

Willow Garage





Intel Haswell

Shortest-path variants

- Single-source: fix source s ∈ V, and find shortest paths from s to every other vertex v ∈ V
- Single-destination: similarly for destination
- Single-pair: given u,v ∈ V, find shortest path
  - No better way known than using single-source
- All-pairs: simultaneously find shortest paths for all possible sources and destinations (ch25)

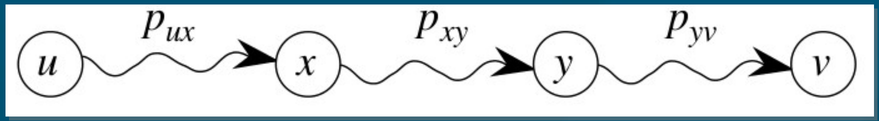
- Negative-weight edges: usually allowable
  - As long as there are no net-negative cycles!
  - Cycles with net weight ≥0 don't help, either

Generic outline of solutions

- Output: for each vertex $v \in V$,
  - v.d: shortest-path estimate
    - Initialised to ∞ (except s.d=0). Always ≥ δ(s,v), and progressively reduced until v.d = δ(s,v) at solution
  - v.parent: links form shortest-path tree
- Edge relaxation: can we improve the shortest-path estimate for v by using the edge (u,v)?
    - Relax(u, v, w):
      - if v.d > u.d + w(u,v):
        - v.d = u.d + w(u,v)
        - v.parent = u
- All our single-source shortest-path algorithms start by initialising v.d, v.parent, then relaxing edges

Optimal substructure

- Any subpath of a shortest path is a shortest path:
  - Cut-and-paste proof:



  - Let $p = p_{ux} + p_{xy} + p_{yv}$ be a shortest path u→v:
    - $\delta(u,v) = w(p) = w(p_{ux}) + w(p_{xy}) + w(p_{yv})$
  - Let p'_{xy} be a shorter path x→y: $w(p'_{xy}) < w(p_{xy})$
  - Then we can build a shorter path p' for u→v:
    - $p' = p_{ux} + p'_{xy} + p_{yv}$
    - $w(p') = w(p_{ux}) + w(p'_{xy}) + w(p_{yv})$
      $< w(p_{ux}) + w(p_{xy}) + w(p_{yv}) = w(p)$
  - This contradicts the assumption that p was the shortest path for u→v

Properties / lemmas

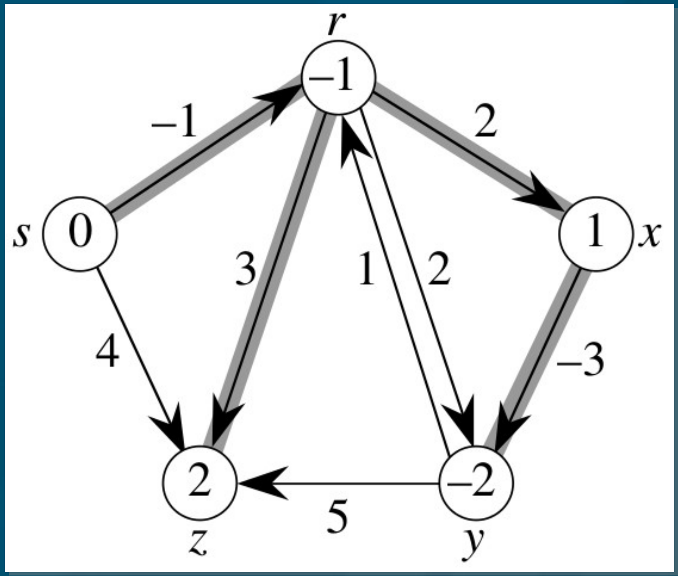
- Triangle inequality: $\delta(s,v) \leq \delta(s,u) + w(u,v)$
- Upper-bound property: $v.d \geq \delta(s,v)$ always, and once $v.d = \delta(s,v)$, it never increases again
  - Relaxing an edge can only lower $v.d$
- No-path property: if $\delta(s,v)=\infty$, then $v.d=\infty$ always
- Convergence property: if $s \rightsquigarrow u \to v$ is a shortest path with $u.d = \delta(s,u)$, then after Relax(u,v,w), we will have $v.d = \delta(s,v)$
  - By optimal substruct: $\delta(s,u) + w(u,v) = \delta(s,v)$
- Path relaxation property: if $p = (v_0=s, \ldots, v_k)$ is a shortest path to v_k, then after relaxing its edges in order: $v_k.d = \delta(s,v_k)$

Outline for today

- Minimum spanning tree
  - Generic solution
  - Kruskal's algorithm
  - Prim's algorithm
  - Uniqueness of MST
- Single-source shortest paths
  - Properties / lemmas about shortest paths
  - Bellman-Ford algorithm
  - Special case if no cycles (DAG)
  - Dijkstra's algorithm

Bellman-Ford algorithm

- Allows negative-weight edges
  - Returns FALSE if net-negative cycle is reachable
- Relax every edge, |V|-1 times:
  - BellmanFord(V, E, w, s):
    - InitSingleSource(V, E, s)
    - for i = 1 to |V|-1:
      - for each (u,v) ∈ E:
        - Relax(u,v,w)
    - for each (u,v) ∈ E:
      - if v.d > u.d + w(u,v)
        - return FALSE

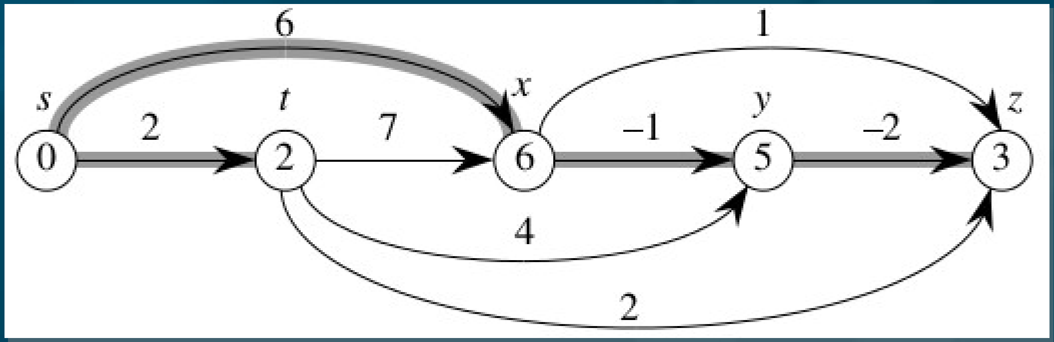


Run time: Θ(VE)

- Convergence: shortest paths have ≤ |V|-1 edges
  - Each iteration relaxes one edge along path $p = (v_0 = s, \ldots, v_k)$, so |V|-1 iterations is enough

Single-source in a DAG

- Directed acyclic graph: no worries about cycles
- Pre-sort vertices by topological sort:
  - For all paths, edges are relaxed in order
  - Don't need to iterate |V|-1 times over edges
    - ShortestPathDAG(V, E, w, s):
      - TopologicalSort(V, E)
      - InitSingleSource(V, E, s)
      - for each u ∊ V in topo order:
        - for each v ∊ Adj(u):
          - Relax(u,v,w)



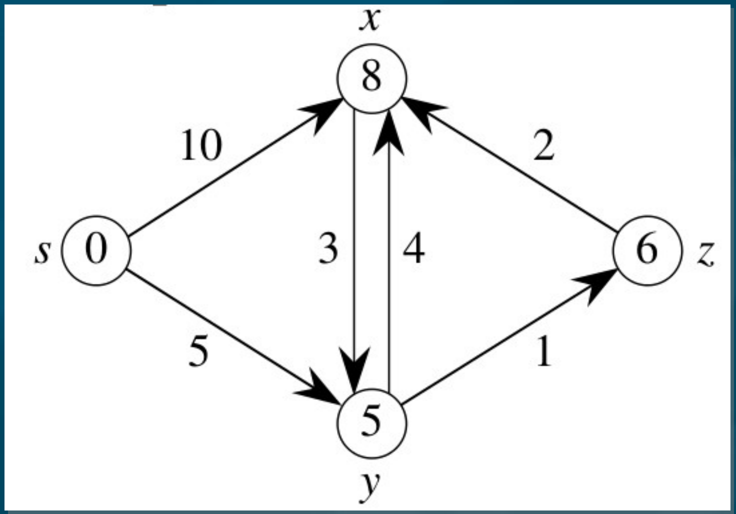
- Time: Θ(V+E)!

Outline for today

- Minimum spanning tree
  - Generic solution
  - Kruskal's algorithm
  - Prim's algorithm
  - Uniqueness of MST
- Single-source shortest paths
  - Properties / lemmas about shortest paths
  - Bellman-Ford algorithm
  - Special case if no cycles (DAG)
  - Dijkstra's algorithm

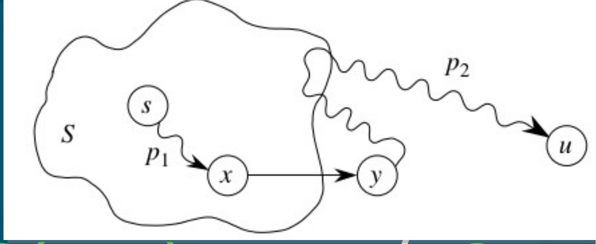
Dijkstra's algorithm

- No negative-weight edges allowed
- Weighted version of breadth-first search
- Use priority queue instead of FIFO
  - Keys are the shortest-path estimates v.d
  - Similar to Prim's algo but calculating v.d
    - Dijkstra(V, E, w, s):
      - InitSingleSource(V, E, s)
      - Q = new PriorityQueue(V)
      - while Q not empty:
        - u = ExtractMin(Q)
        - for each v ∈ Adj(u):
          - Relax(u,v,w)



- Greedy choice: select u with lowest u.d

Dijkstra: correctness



- Loop invariant: at top of loop, u.d = δ(s,u) ∀ u ∉ Q
- Proof: suppose not: let u be the first vertex removed from Q that has u.d ≠ δ(s,u)
  - ∃ path s ↝ u (otherwise, u.d = ∞ = δ(s,u))
  - Let p be a shortest path s ↝ u, and let (x,y) be the first edge in p crossing from !Q to Q
    - Then x.d = δ(s,d) (as u is first to have u.d ≠ δ(s,u))
  - (x,y) was then relaxed, so y.d = δ(s,y) (convgc)
    - y on shortest path, so δ(s,y) ≤ δ(s,u) ≤ u.d
  - But both y,u ∈ Q when ExtractMin, so u.d ≤ y.d
  - Hence y.d = u.d, so u.d = δ(s,u), contradiction

Dijkstra: running time

- Init for weights and Q takes Θ(V)
- ExtractMin is run exactly |V| times
- DecreaseKey (called by Relax) is run O(E) times
- Using binary max-heaps:
  - All operations are O(lg V)
  - ⇒ Total time O(E lg V)
- Using Fibonacci heaps:
  - ExtractMin takes O(1) amortised time
  - Other operations total O(V) ops with amortised time O(lg V) each
  - ⇒ Total time O(V lg V + E)