

# CSE 421 Algorithms

Autumn 2019  
Lecture 10  
Minimum Spanning Trees

Edge costs are assumed to be non-negative

## Dijkstra's Algorithm Implementation and Runtime

$S = \{ \}; \quad d[s] = 0; \quad d[v] = \text{infinity for } v \neq s$

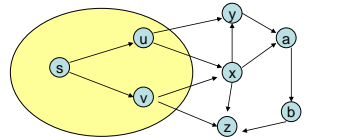
While  $S \neq V$

    Choose  $v$  in  $V-S$  with minimum  $d[v]$

    Add  $v$  to  $S$

    For each  $w$  in the neighborhood of  $v$

$d[w] = \min(d[w], d[v] + c(v, w))$



HEAP OPERATIONS

$n$  Extract Mins

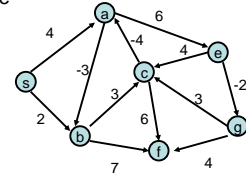
$m$  Heap Updates

## Run Time

- Basic Heap Implementation
  - $O(\log n)$  extract min and update key
  - $O((m + n) \log n)$  run time
- Fancy data structures: Fibonacci Heaps
  - $O(m + n \log n)$
- Dense graphs
  - $O(n^2)$

## Shortest Paths

- Negative Cost Edges
  - Dijkstra's algorithm assumes positive cost edges
  - For some applications, negative cost edges make sense
  - Shortest path not well defined if a graph has a negative cost cycle

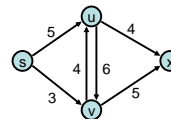


## Negative Cost Edge Preview

- Topological Sort can be used for solving the shortest path problem in directed acyclic graphs
- Bellman-Ford algorithm finds shortest paths in a graph with negative cost edges (or reports the existence of a negative cost cycle).

## Bottleneck Shortest Path

- Define the bottleneck distance for a path to be the maximum cost edge along the path

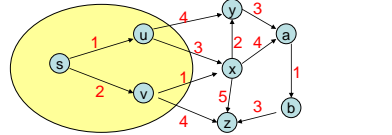


The figure consists of two parts. The left part is a weighted undirected graph with 8 nodes labeled a, b, c, d, e, f, g, and s. The edges and their weights are: (a,b) weight 4, (a,c) weight 4, (a,d) weight 6, (b,c) weight 3, (b,f) weight 7, (c,d) weight 5, (c,e) weight 4, (c,f) weight 5, (c,g) weight 3, (d,e) weight 6, and (e,g) weight 2. The right part shows the same 8 nodes arranged in a 2D space, representing a node layout. The nodes are approximately at the following relative positions: s is on the far left; a, b, c, d, e, f, and g are clustered on the right, with d at the top and f at the bottom.

$S = \{ \}; \quad d[s] = \text{negative infinity}; \quad d[v] = \text{infinity for } v \neq s$

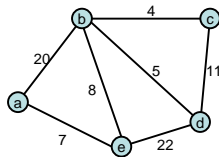
Choose  $v$  in  $V-S$  with minimum  $d[v]$

For each  $w$  in the neighborhood of  $v$

$$d[w] = \min(d[w], \max(d[v], c(v, w)))$$


- Introduce Problem
- Demonstrate three different greedy algorithms
- Provide proofs that the algorithms work

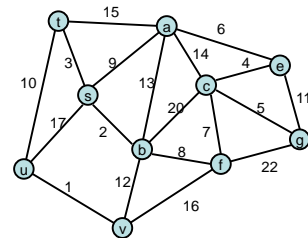
- Extend a tree by including the cheapest out going edge
- Add the cheapest edge that joins disjoint components
- Delete the most expensive edge that does not disconnect the graph



- Extend a tree by including the cheapest out going edge

Construct the MST with Prim's algorithm starting from vertex a

Label the edges in order of insertion

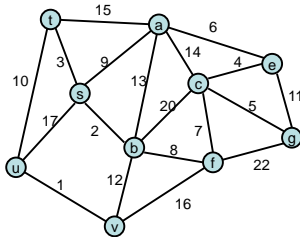


## Greedy Algorithm 2 Kruskal's Algorithm

- Add the cheapest edge that joins disjoint components

Construct the MST  
with Kruskal's  
algorithm

Label the edges in  
order of insertion

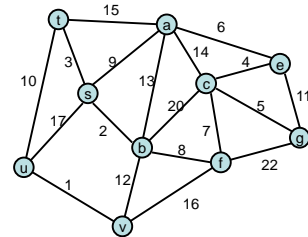


## Greedy Algorithm 3 Reverse-Delete Algorithm

- Delete the most expensive edge that does not disconnect the graph

Construct the MST  
with the reverse-  
delete algorithm

Label the edges in  
order of removal



## Dijkstra's Algorithm for Minimum Spanning Trees

$S = \{ \}$ ;  $d[s] = 0$ ;  $d[v] = \text{infinity}$  for  $v \neq s$

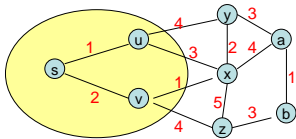
While  $S \neq V$

Choose  $v$  in  $V-S$  with minimum  $d[v]$

Add  $v$  to  $S$

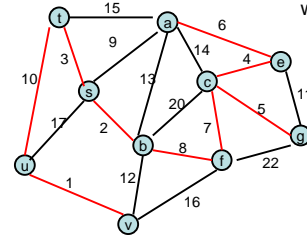
For each  $w$  in the neighborhood of  $v$

$d[w] = \min(d[w], c(v, w))$



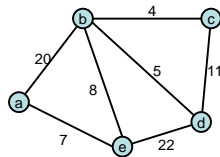
## Minimum Spanning Tree

Undirected Graph  
 $G=(V,E)$  with edge  
weights



## Greedy Algorithms for Minimum Spanning Tree

- **[Prim]** Extend a tree by including the cheapest outgoing edge
- **[Kruskal]** Add the cheapest edge that joins disjoint components
- **[ReverseDelete]** Delete the most expensive edge that does not disconnect the graph

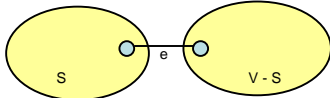


## Why do the greedy algorithms work?

- For simplicity, assume all edge costs are distinct

## Edge inclusion lemma

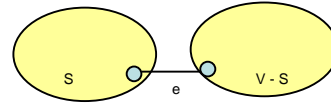
- Let  $S$  be a subset of  $V$ , and suppose  $e = (u, v)$  is the minimum cost edge of  $E$ , with  $u$  in  $S$  and  $v$  in  $V-S$
- $e$  is in every minimum spanning tree of  $G$ 
  - Or equivalently, if  $e$  is not in  $T$ , then  $T$  is not a minimum spanning tree



$e$  is the minimum cost edge  
between  $S$  and  $V-S$

## Proof

- Suppose  $T$  is a spanning tree that does not contain  $e$
- Add  $e$  to  $T$ , this creates a cycle
- The cycle must have some edge  $e_1 = (u_1, v_1)$  with  $u_1$  in  $S$  and  $v_1$  in  $V-S$



- $T_1 = T - \{e_1\} + \{e\}$  is a spanning tree with lower cost
- Hence,  $T$  is not a minimum spanning tree