# CSE 421
# Introduction to Algorithms

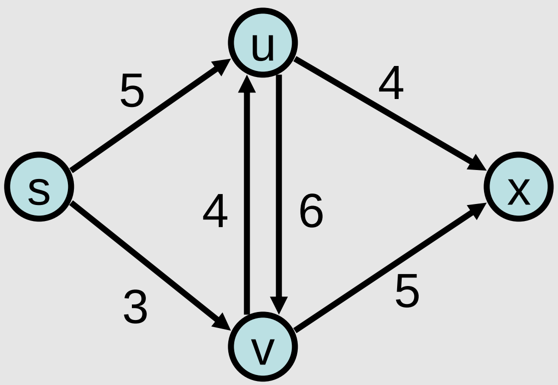
Winter 2024

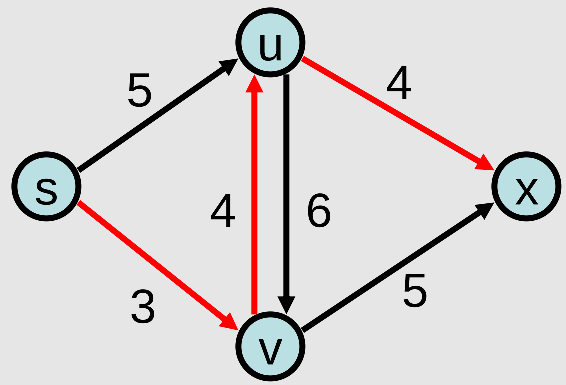
Lecture 8

Minimum Spanning Trees

# Bottleneck Shortest Path

- Define the bottleneck distance for a path P, $Len_B(P)$ to be the maximum cost edge along the path




$Len_B(x,y) = \text{Min } \{P \text{ from } x \text{ to } y \mid Len_B(P)\ \}$

# Dijkstra's Algorithm for Bottleneck Shortest Paths

S = { }; d[s] = negative infinity; d[v] = infinity for v != s

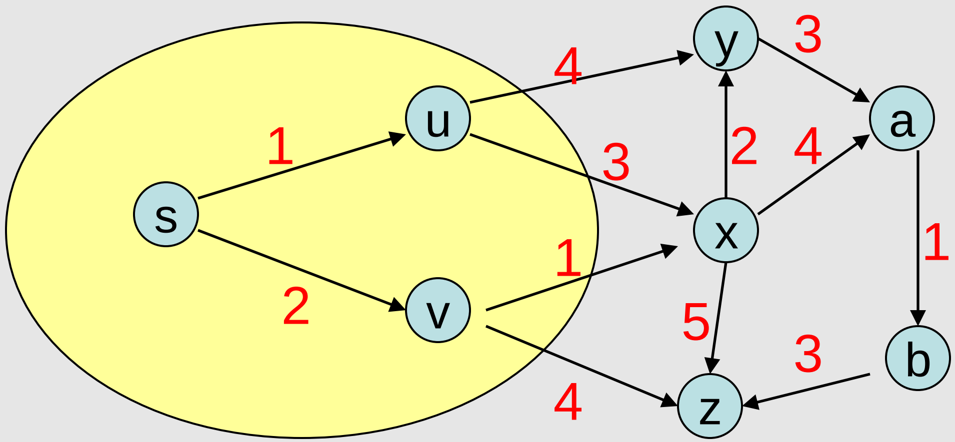
While S != V

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

Add v to S

For each w in the neighborhood of v

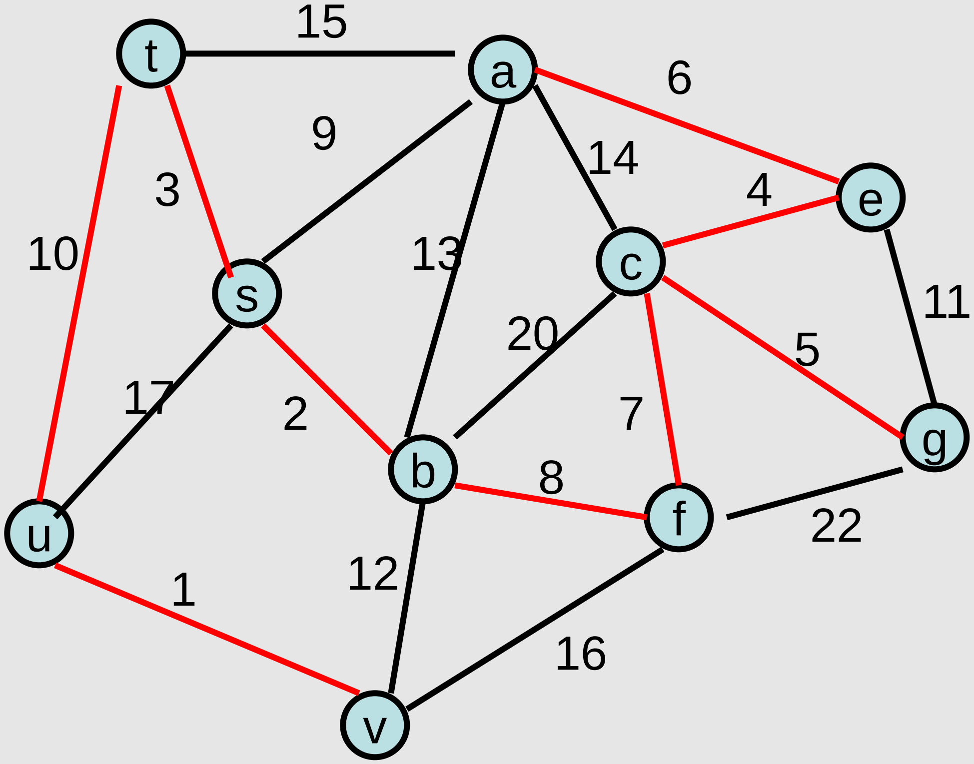
d[w] = min(d[w], max(d[v], c(v, w)))



# Minimum Spanning Tree

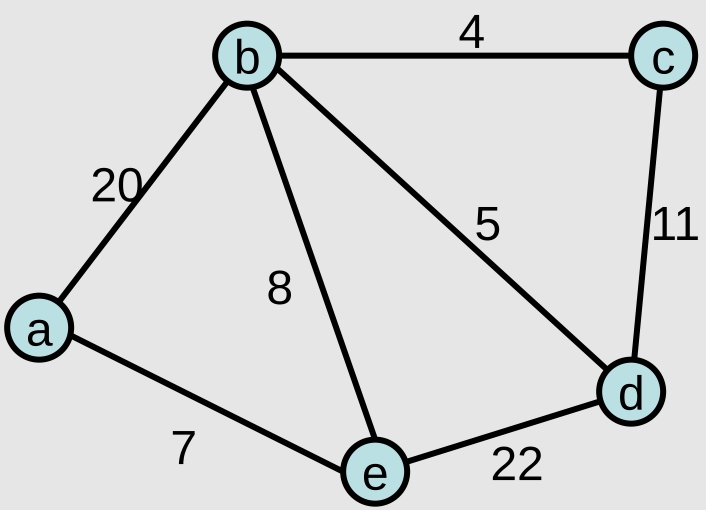
- Introduce Problem
- Demonstrate Prim’s and Kruskal’s algorithms
- Provide proofs that the algorithms work

# Minimum Spanning Tree



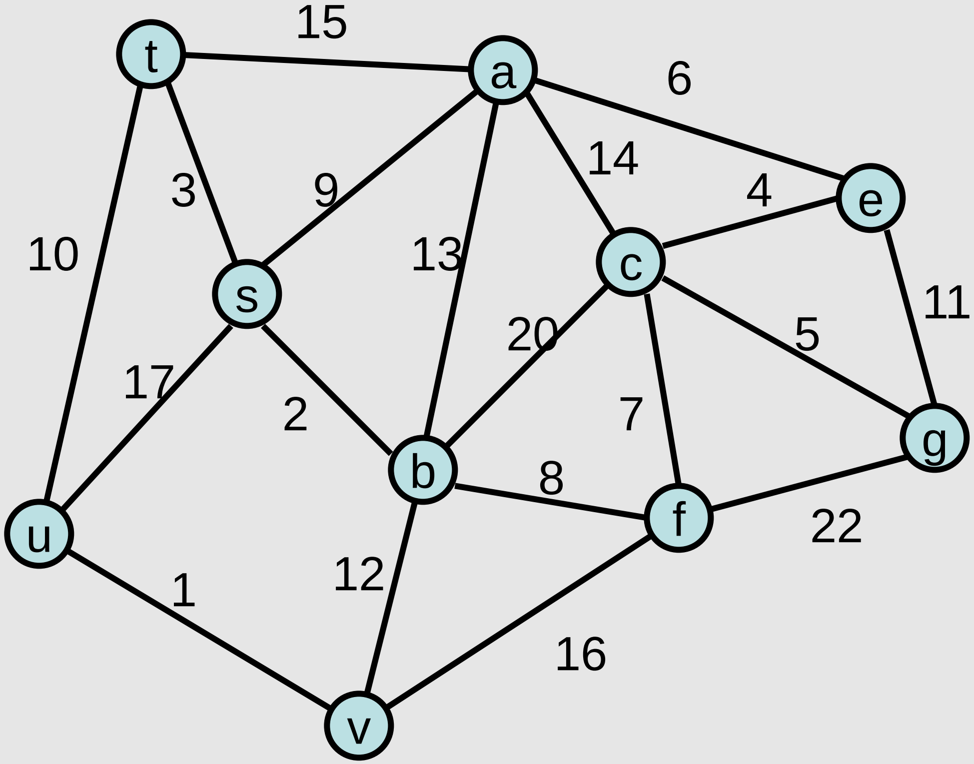
# Greedy Algorithms for Minimum Spanning Tree

- 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~~



# Greedy Algorithm 1
# Prim's Algorithm

- Extend a tree by including the cheapest out going edge

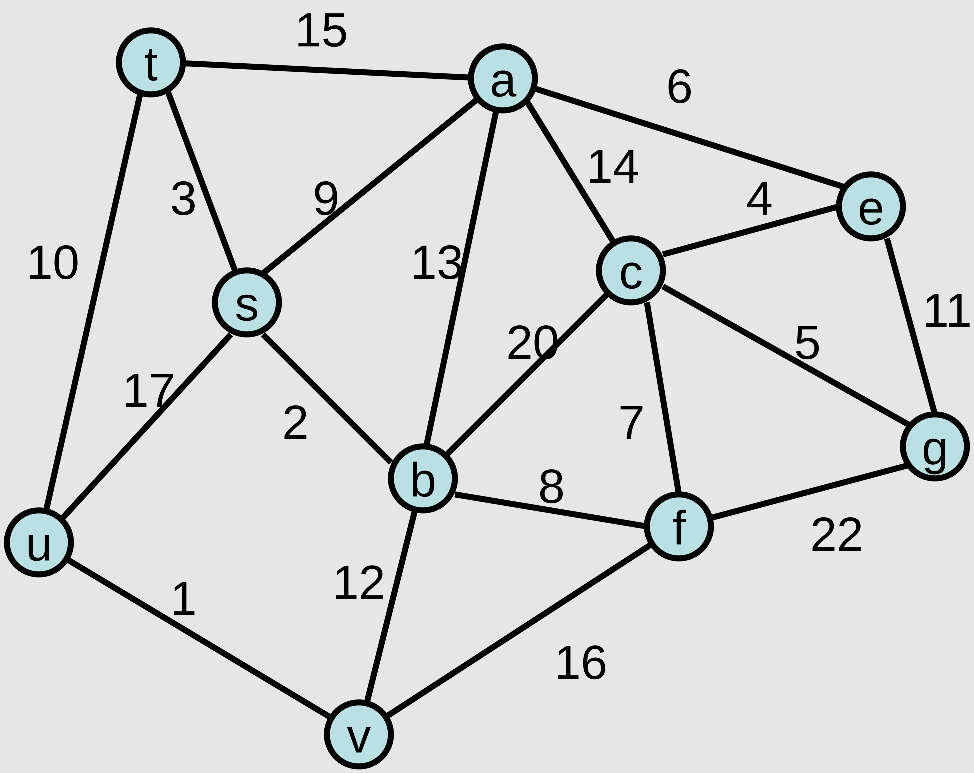



Construct the MST with Prim's algorithm starting from vertex a

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

# Dijkstra's Algorithm for Minimum Spanning Trees

S = { }; d[s] = 0; d[v] = infinity for v != s

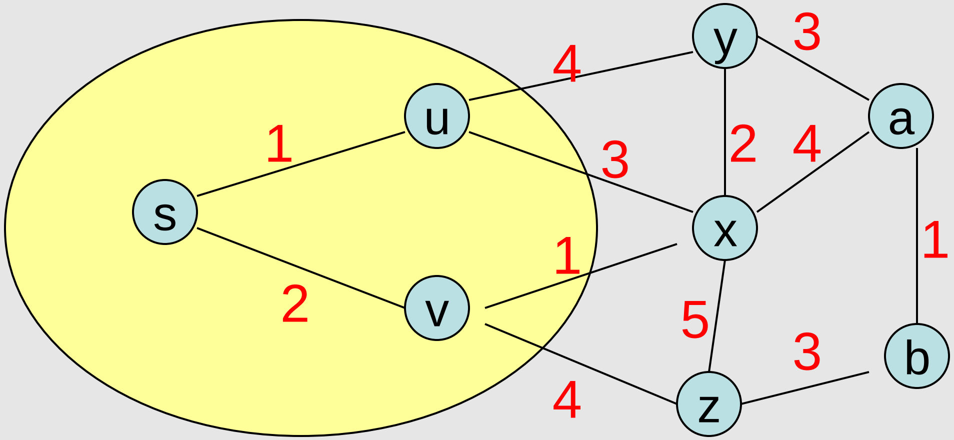
While S != V

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

Add v to S

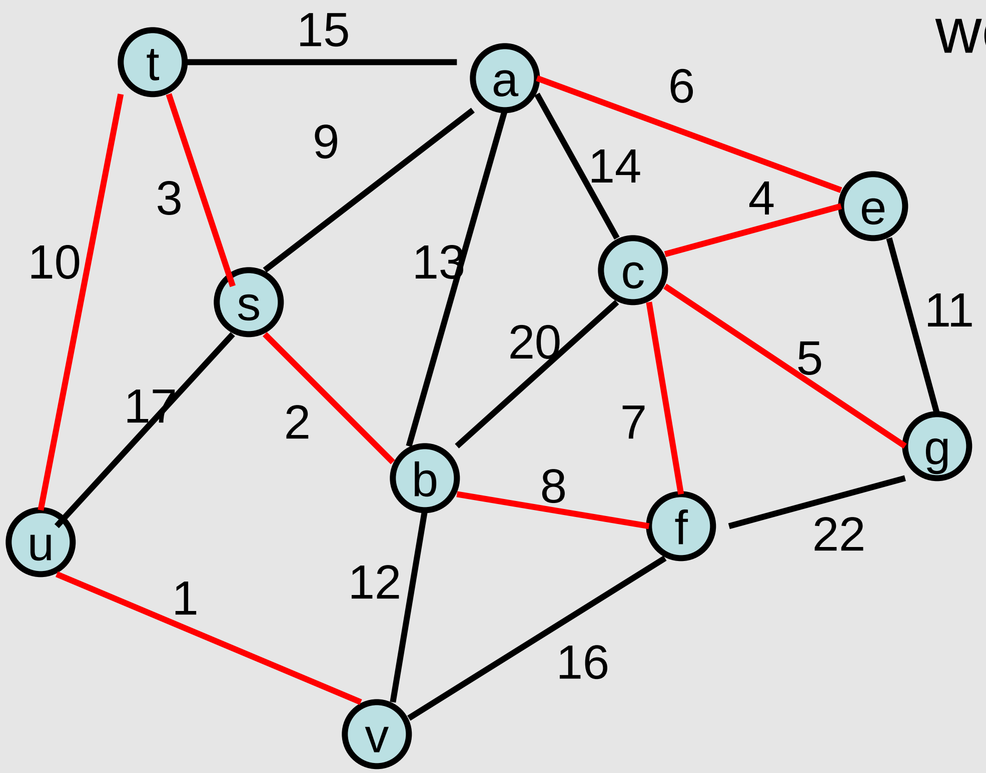
For each w in the neighborhood of v

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



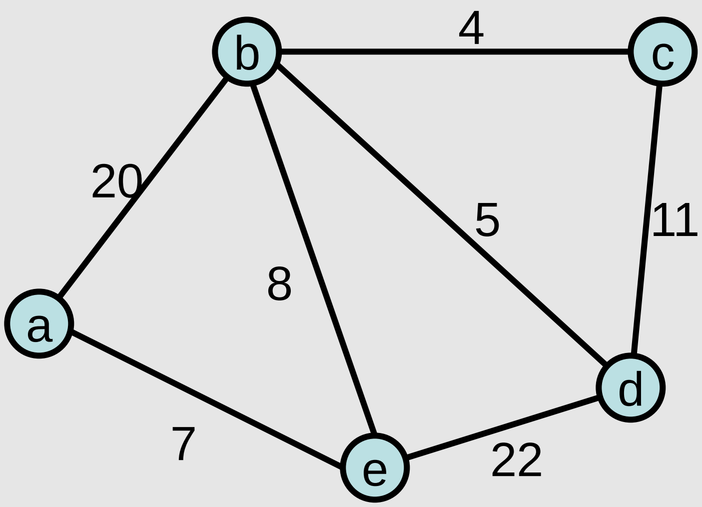
# Minimum Spanning Tree

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



# Greedy Algorithms for Minimum Spanning Tree

- [Prim] Extend a tree by including the cheapest out going edge
- [Kruskal] Add the cheapest edge that joins disjoint components

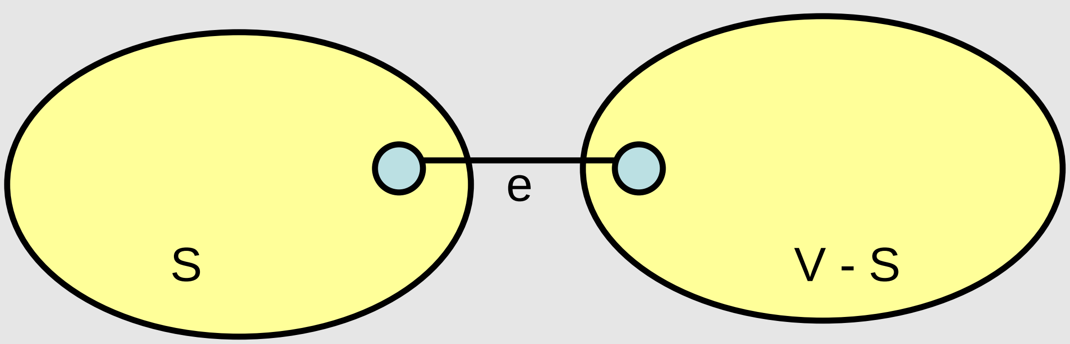


# Why do the greedy algorithms work?

- For simplicity, assume all edge costs are distinct

# Edge inclusion lemma

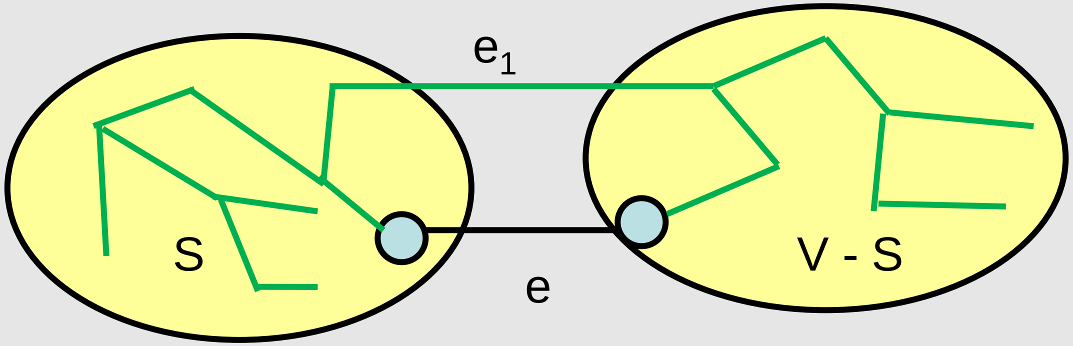
- Let S be a subset of V, and suppose e = (u, v) is the minimum cost edge of E, with u in S and v in V-S
- e is in every minimum spanning tree of G
  - Or equivalently, if e is not in T, then T is not a minimum spanning tree



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

# Optimality Proofs

- Prim’s Algorithm computes a MST
- Kruskal’s Algorithm computes a MST

- Show that when an edge is added to the MST by Prim or Kruskal, the edge is the minimum cost edge between S and V-S for some set S.

# Prim’s Algorithm

S = { }; T = { };

while S != V

choose the minimum cost edge
e = (u,v), with u in S, and v in V-S

add e to T

add v to S

# Prove Prim's algorithm computes an MST

- Show an edge e is in the MST when it is added to T

# Kruskal's Algorithm

Let $C = \{\{v_1\}, \{v_2\}, . . ., \{v_n\}\}$; $T = \{ \}$

while $|C| > 1$

> Let $e = (u, v)$ with $u$ in $C_i$ and $v$ in $C_j$ be the minimum cost edge joining distinct sets in $C$
>
> Replace $C_i$ and $C_j$ by $C_i \cup C_j$
>
> Add $e$ to $T$

# Prove Kruskal’s algorithm computes an MST

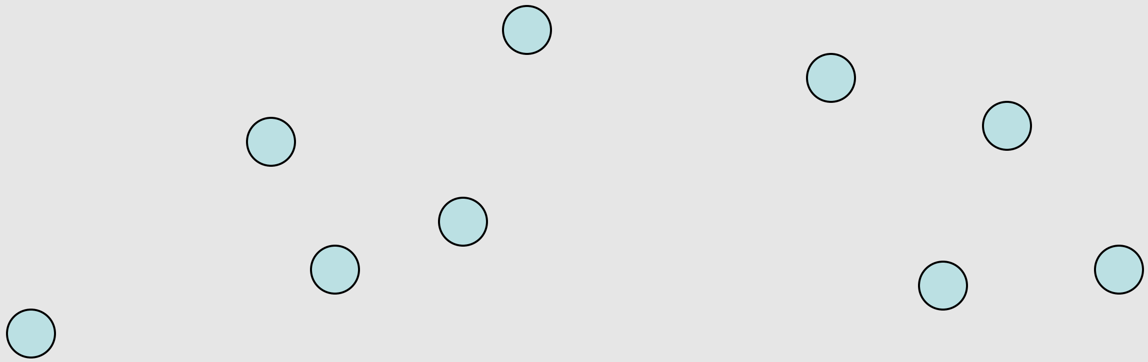
- Show an edge e is in the MST when it is added to T

# Dealing with the assumption of no equal weight edges

- Force the edge weights to be distinct
  - Give a tie breaking rule for equal weight edges

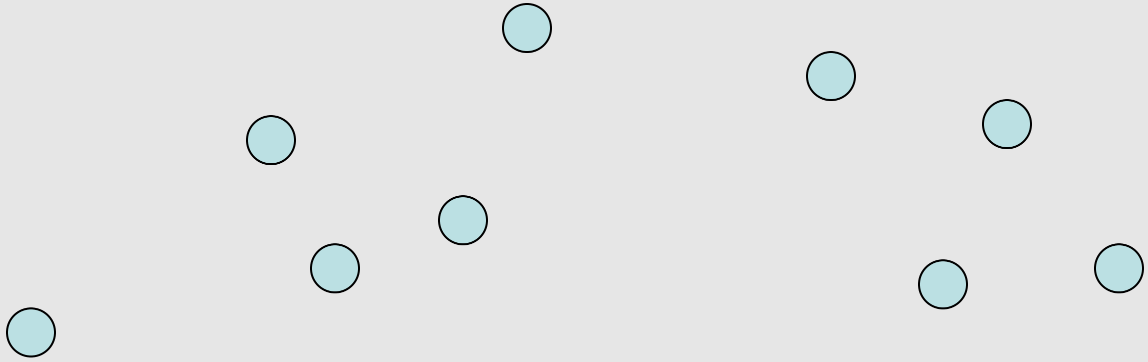
# Application: Clustering

- Given a collection of points in an r-dimensional space and an integer K, divide the points into K sets that are closest together

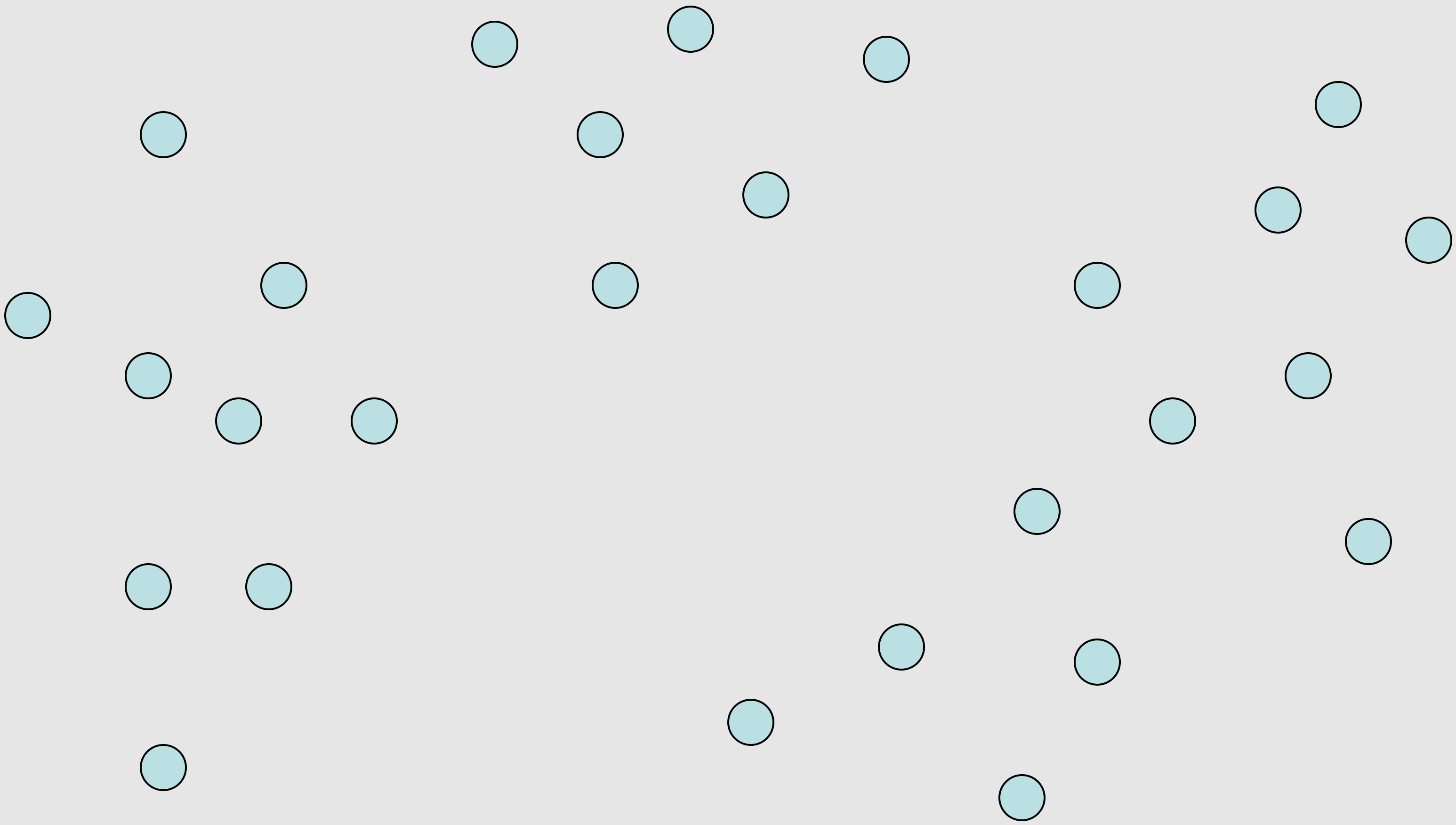


# Distance clustering

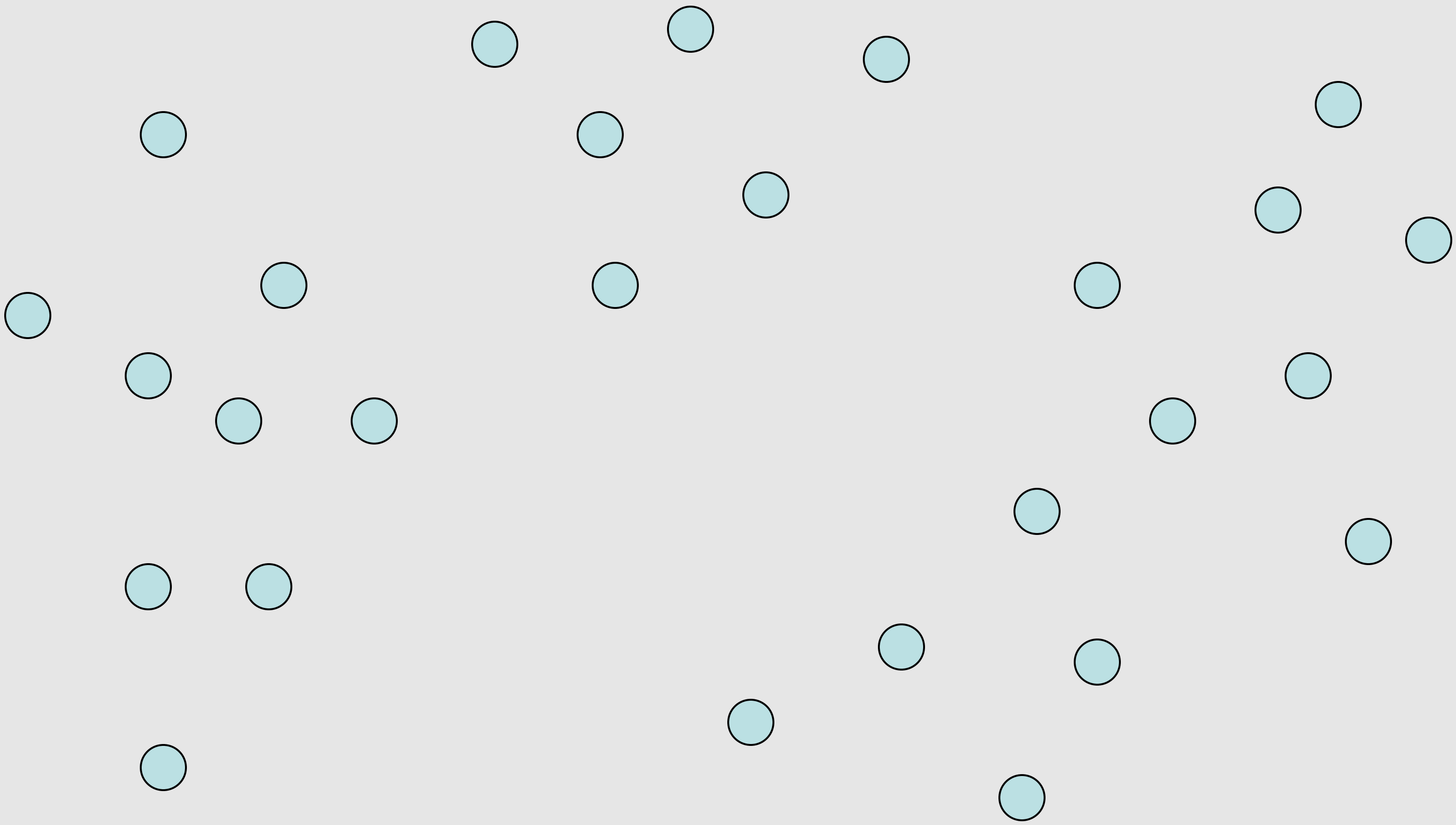
- Divide the data set into K subsets to maximize the distance between any pair of sets
  - dist $(S_1, S_2)$ = min {dist(x, y) | x in $S_1$, y in $S_2$}



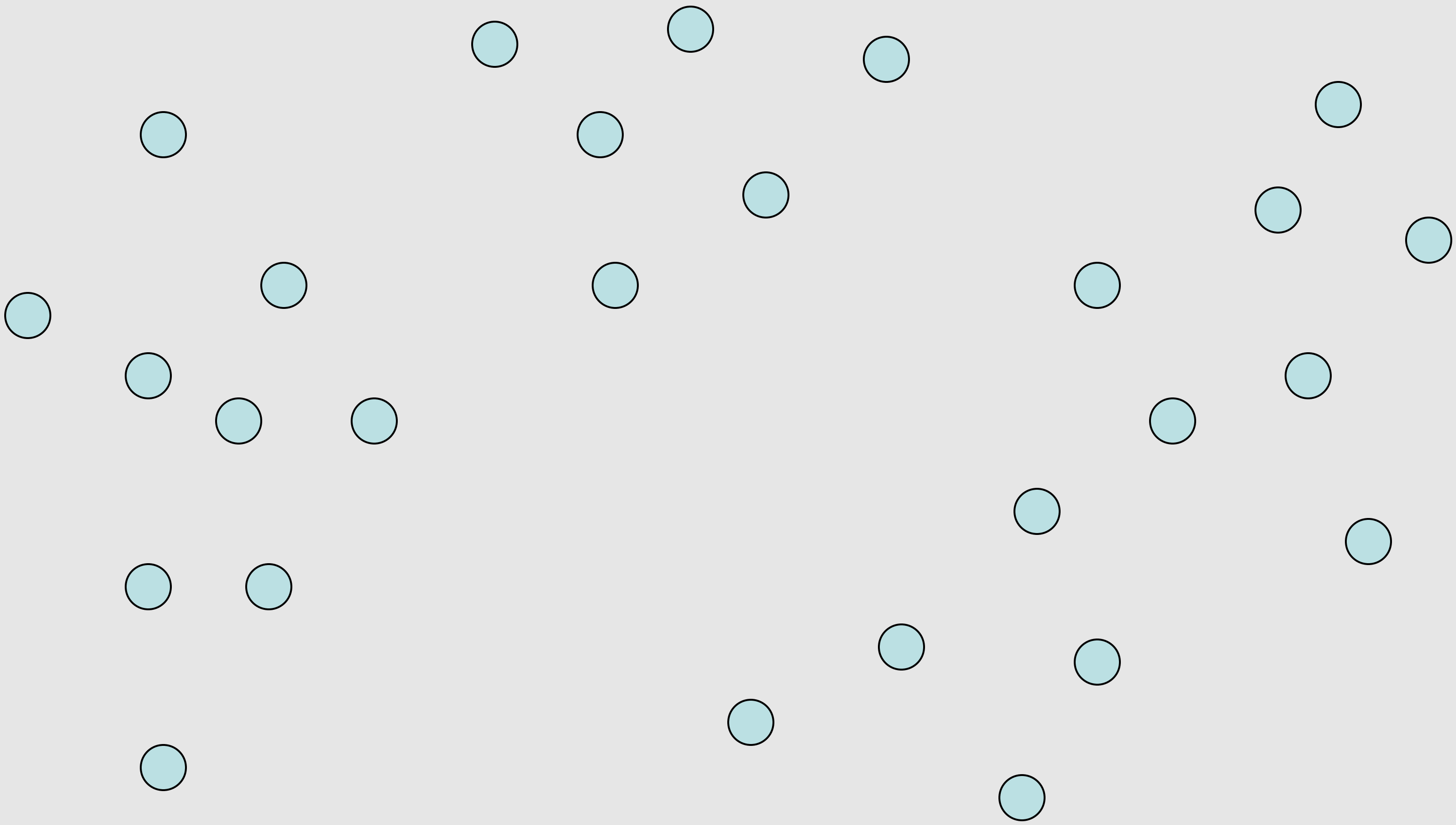
# Divide into 2 clusters



# Divide into 3 clusters



# Divide into 4 clusters



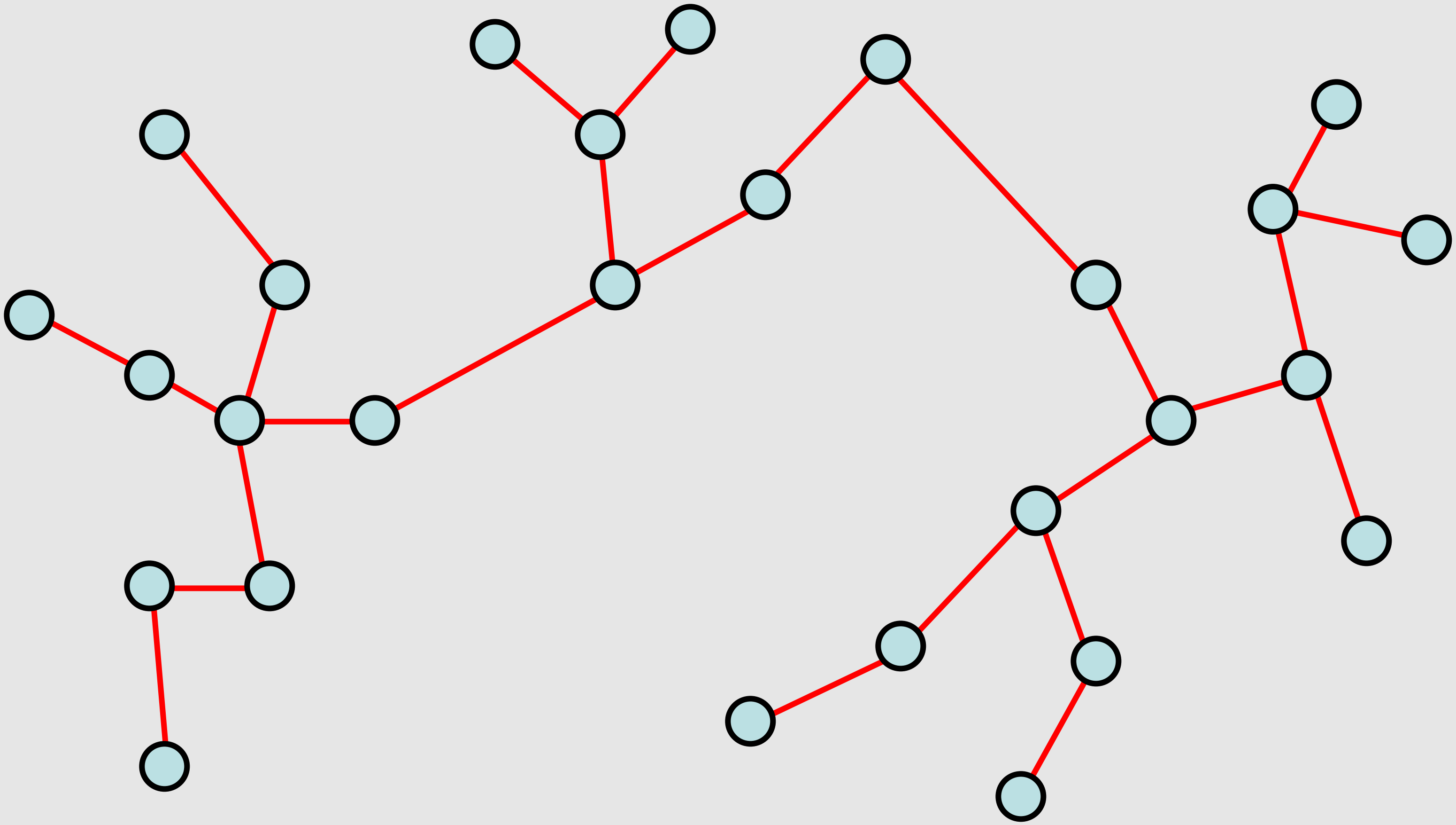
# Distance Clustering Algorithm

Let $C = \{\{v_1\}, \{v_2\}, . . ., \{v_n\}\}$;  $T = \{ \}$

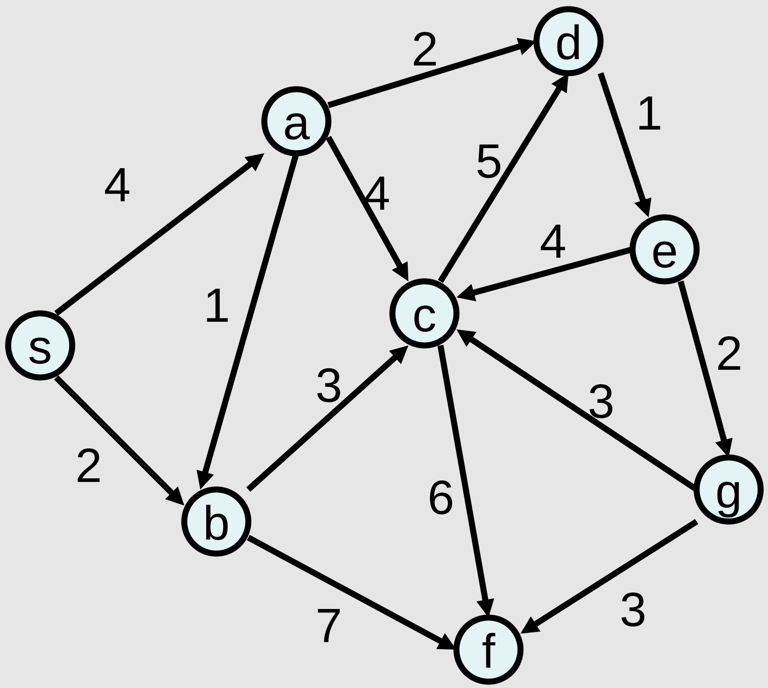
while $|C| > K$

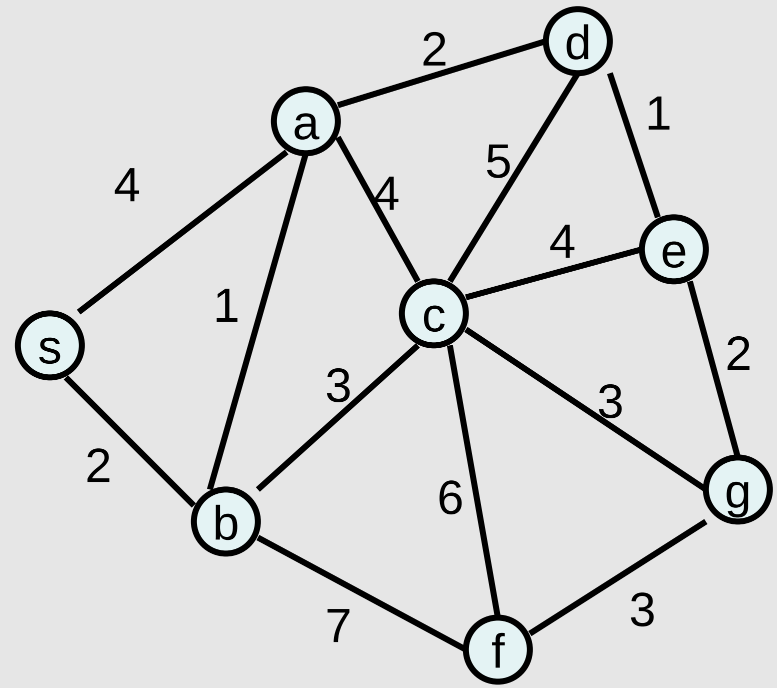
> Let $e = (u, v)$ with $u$ in $C_i$ and $v$ in $C_j$ be the minimum cost edge joining distinct sets in $C$
>
> Replace $C_i$ and $C_j$ by $C_i \cup C_j$

# K-clustering



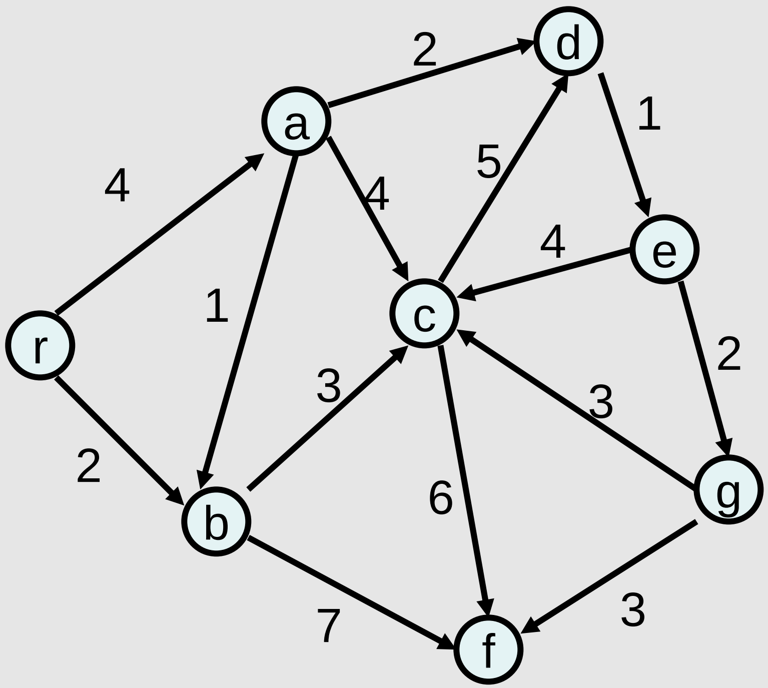
# Shortest paths in directed graphs vs undirected graphs



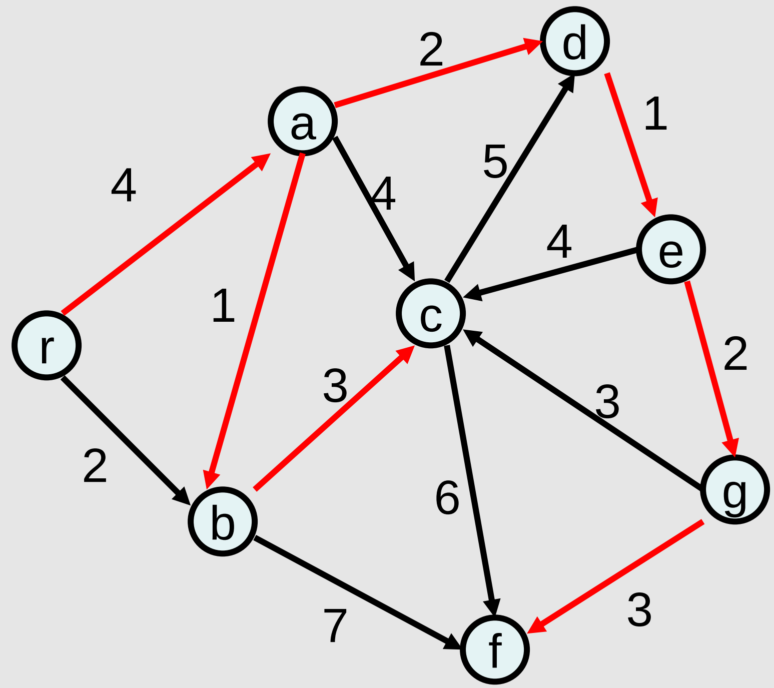



# What about the minimum spanning tree of a directed graph?

- Must specify the root r
- Branching: Out tree with root r

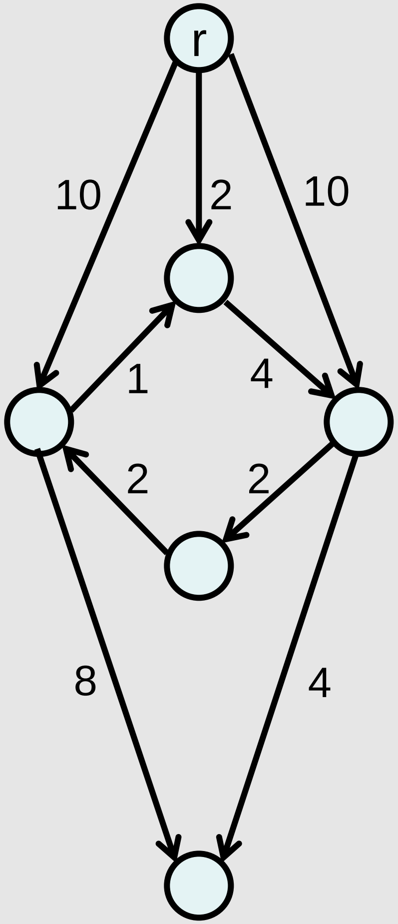


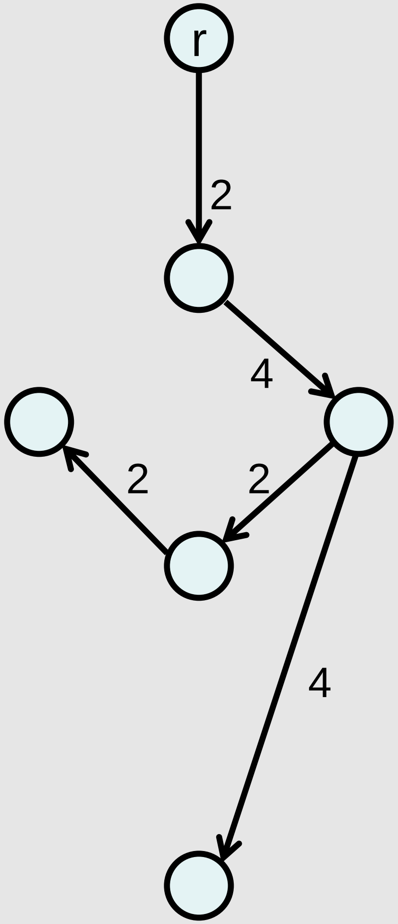

Assume all vertices reachable from r




Also called an arborescence

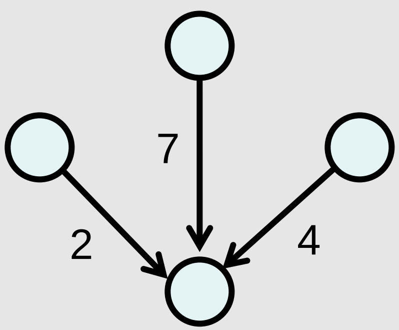
# Finding a minimum branching

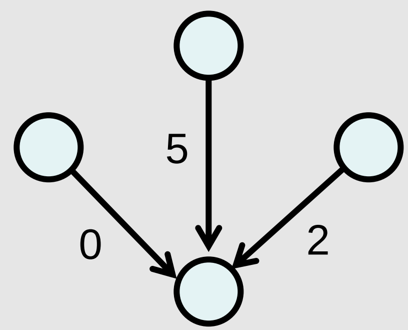


# Finding a minimum branching

- Remove all edges going into r
- Normalize the edge weights, so the minimum weight edge coming into each vertex has weight zero



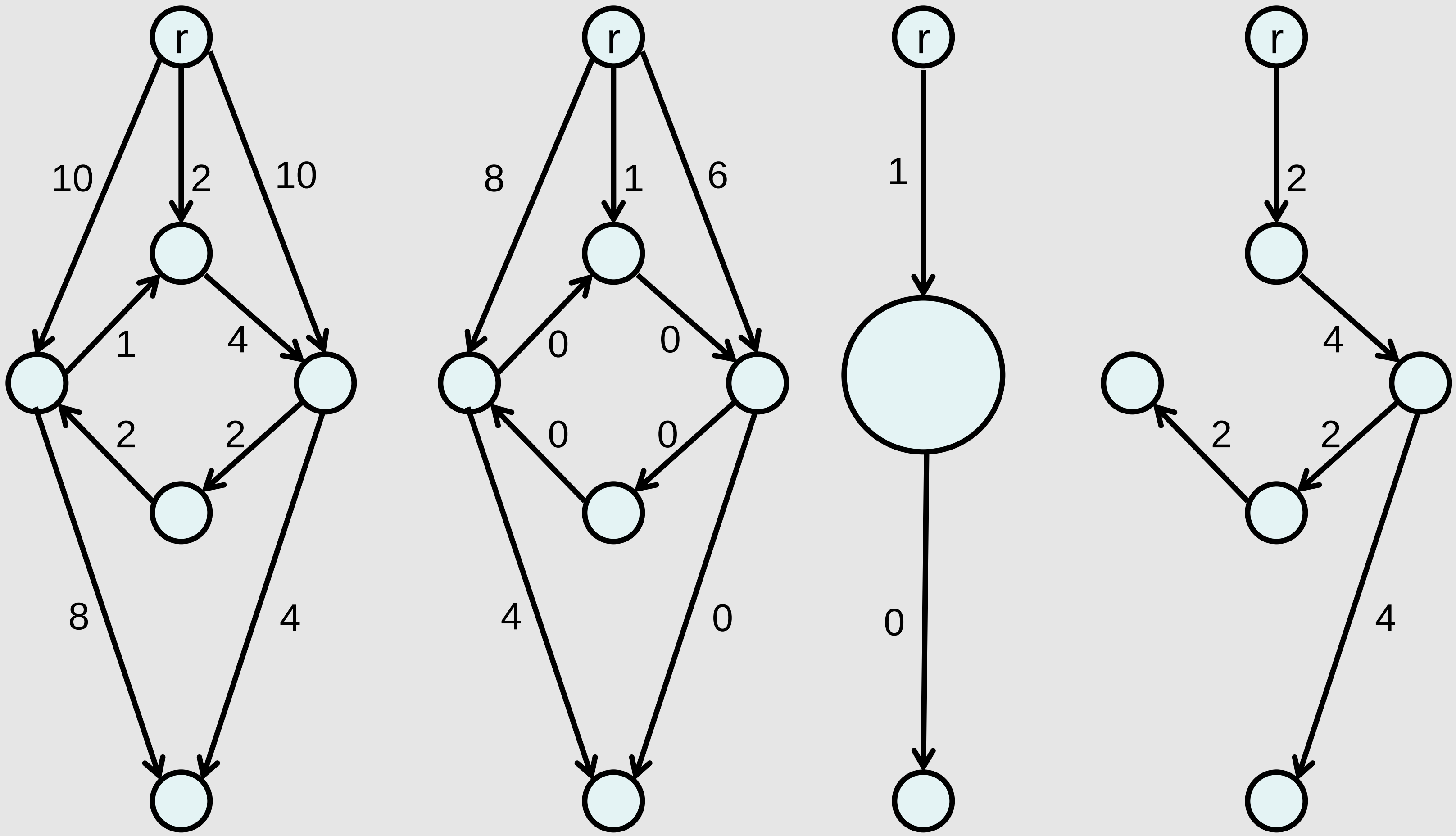




This does not change the edges of the minimum branching

# Finding a minimum branching

- Consider the graph that consists of the minimum cost edge coming in to each vertex
  - If this graph is a branching, then it is the minimum cost branching
  - Otherwise, the graph contains one or more cycles
    - Collapse the cycles in the original graph to super vertices
    - Reweight the graph and repeat the process

# Finding a minimum branching



# Correctness Proof

Lemma 4.38 Let C be a cycle in G consisting of edges of cost 0 with r not in C. There is an optimal branching rooted at r that has exactly one edge entering C.

- The lemma justifies using the edges of the cycle in the branching
- An induction argument is used to cover the multiple levels of compressing cycles