CSE 332: Data Structures and Parallelism

Spring 2022

Richard Anderson

Lecture 25: Minimum Spanning Trees

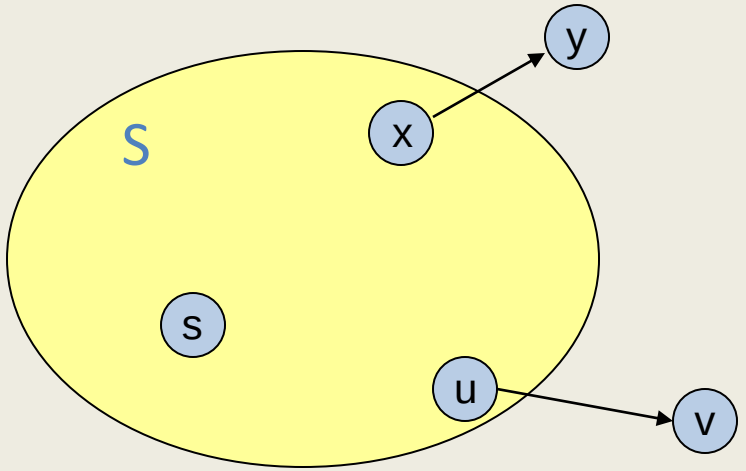
Announcements

- Upcoming lectures
  - Graph Algorithms
    - ~~Graph Traversal~~
    - ~~Shortest Paths~~
    - Minimum Spanning Tree
  - Union-Find Data Structure
  - Theory of NP-Completeness (2 lectures)
- Final Exam, June 9, 8:30-10:20 AM

Assume all edges have non-negative cost

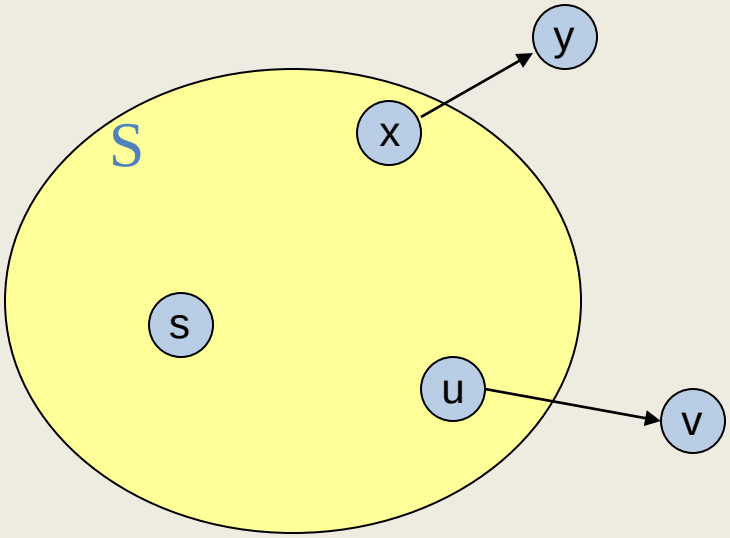
Dijkstra's Algorithm
What about negative cost edges?

```
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
        newCost = d[v] + c(v, w)
        if (newCost < d[w])
            d[w] = newCost
            prev[w] = v
```



Correctness Proof

- Elements in S have the correct label
- Induction: when v is added to S, it has the correct distance label
  - Dist(s, v) = d[v] when v added to S

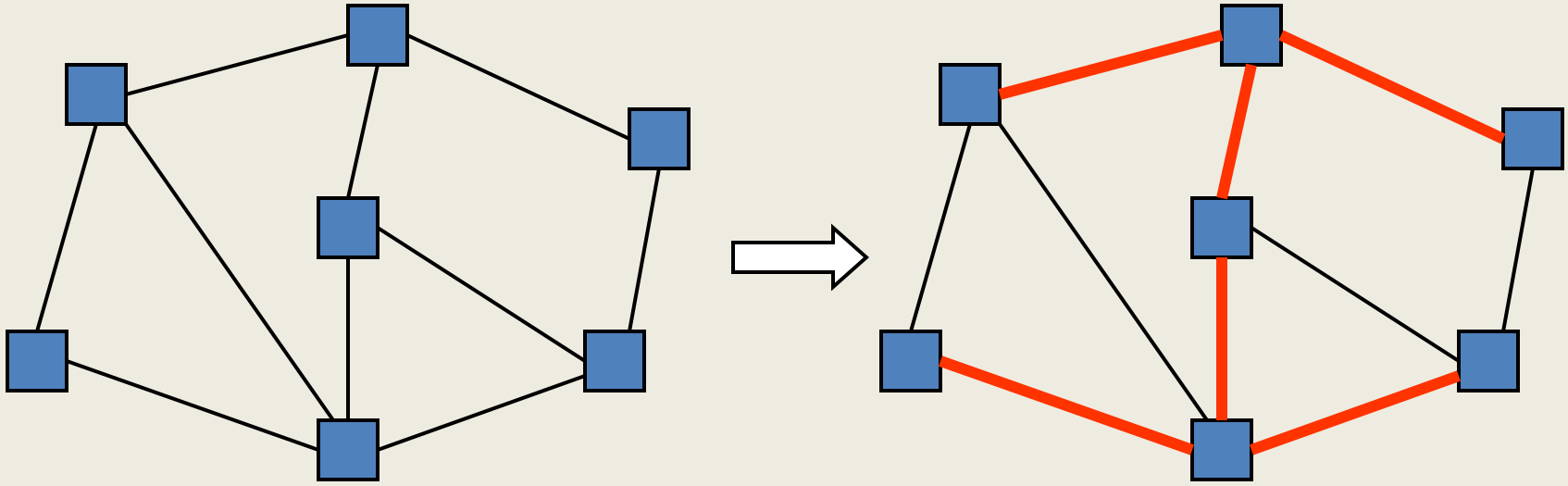


Graph Theory

- G = (V, E)
  - V: vertices, |V| = n
  - E: edges, |E| = m
- Undirected graphs
  - Edges sets of two vertices {u, v}
- Directed graphs
  - Edges ordered pairs (u, v)
- Many other flavors
  - Edge / vertices weights
  - Parallel edges
  - Self loops

- Path: $v_1, v_2, \ldots, v_k$, with (v_i, v_{i+1}) in E
  - Simple Path
  - Cycle
  - Simple Cycle
- Neighborhood
  - N(v)
- Distance
- Connectivity
  - Undirected
  - Directed (strong connectivity)
- Trees
  - Rooted
  - Unrooted

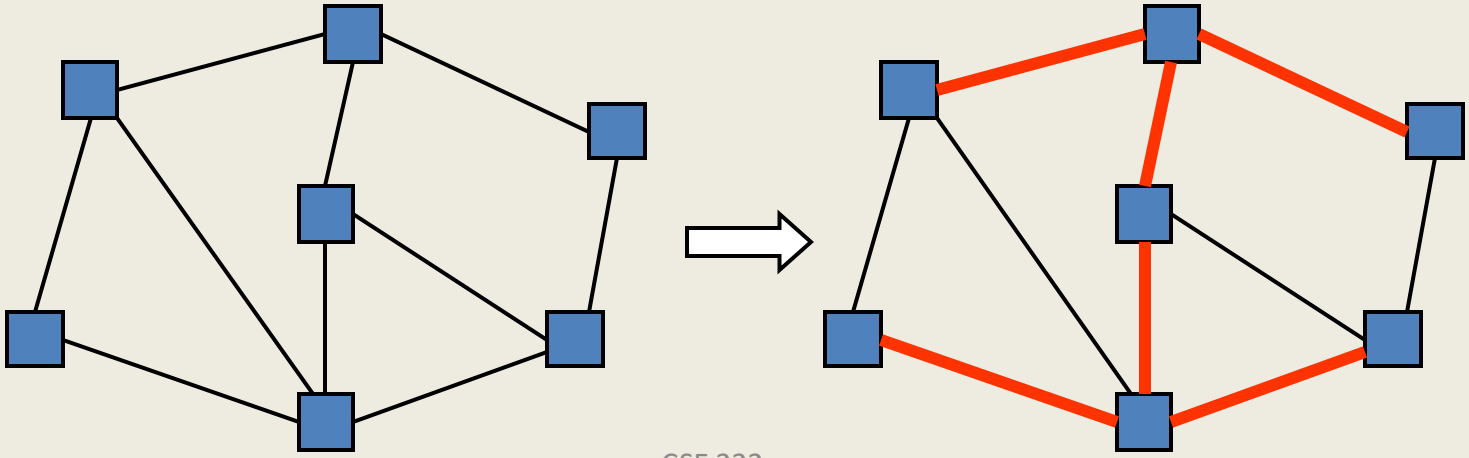
Spanning Tree in an Undirected Graph



Spanning tree
- Connects all the vertices
- No cycles

Spanning Tree Problem

- Input: An undirected graph G = (V,E). G is connected.
- Output: **T** $\subset$ E such that
  - (V,**T**) is a connected graph
  - (V,**T**) has no cycles



Spanning Tree Algorithm

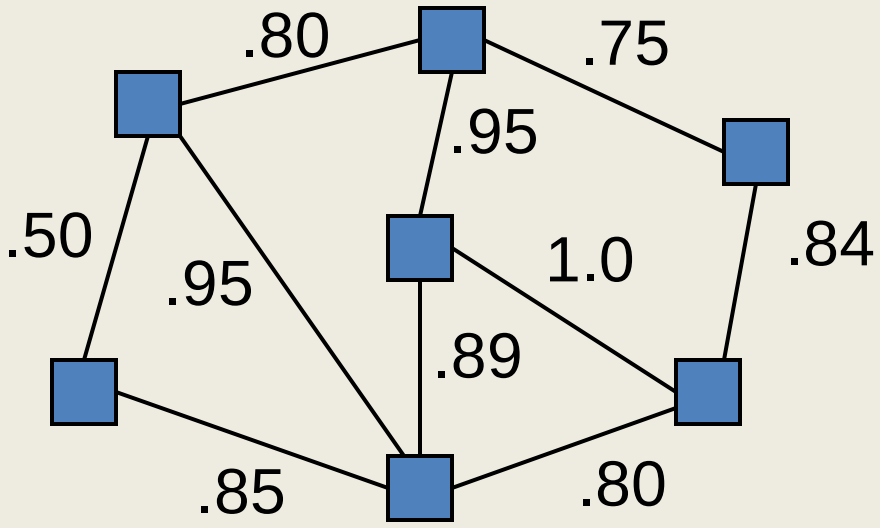
```
ST(Vertex i) {
    mark i;
    for each j adjacent to i  {
        if (j is unmarked) {
           Add (i,j) to T;
           ST(j);
        }
    }
}
```

```
Main( ) {
  T = empty set;
  ST(1);
}
```

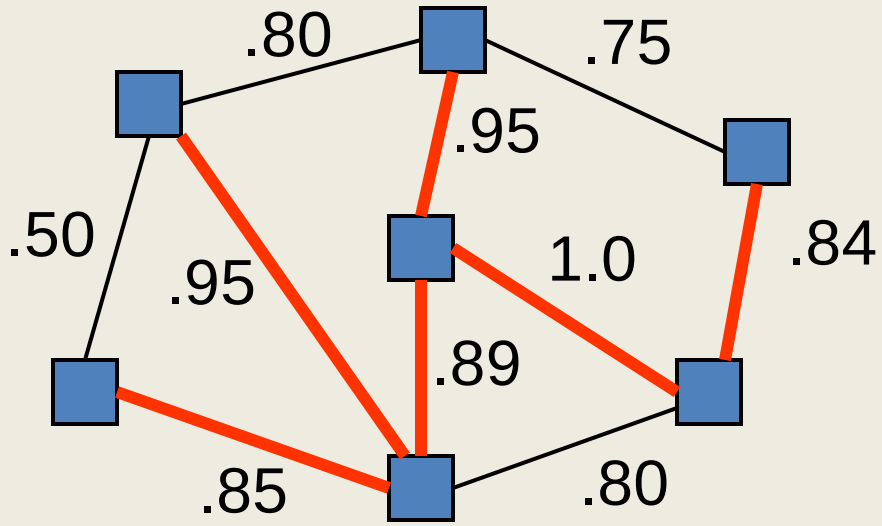
Best Spanning Tree

Finding a reliable routing subnetwork:

- edge cost = probability that it won't fail
- Find the spanning tree that is least likely to fail



Example of a Spanning Tree



Probability of success = .85 x .95 x .89 x .95 x 1.0 x .84
= .5735

Minimum Spanning Trees

Given an undirected graph **G**=(**V,E**), find a graph **G'=(V, E')** such that:

- E' is a subset of E
- |E'| = |V| - 1
- G' is connected
- $\sum_{(u,v)\in E'} c_{uv}$ is minimal

G' is a **minimum spanning tree**.

Minimum Spanning Tree Problem

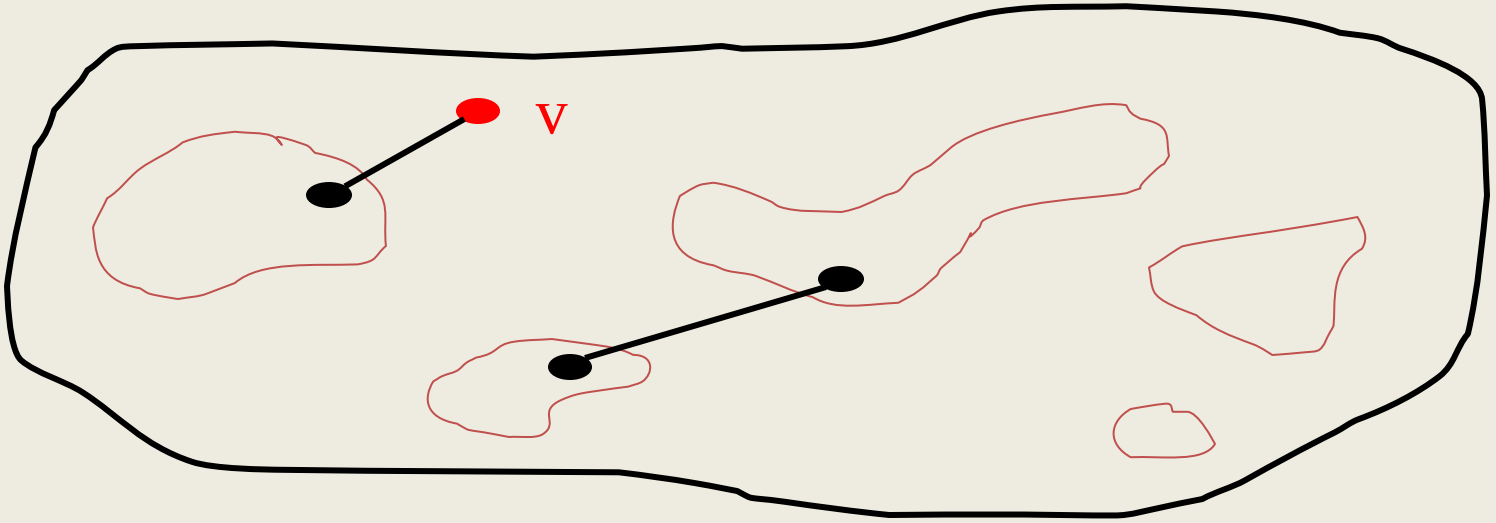
- Input: Undirected Graph G = (V,E) and C(e) is the cost of edge e.
- Output: A spanning tree T with minimum total cost. Find a tree T that minimizes

$$C(T) = \sum_{e \in T} C(e)$$

Kruskal's MST Algorithm

Idea: Grow a forest out of edges that do not create a cycle.
Pick an edge with the smallest weight.

G=(V,E)



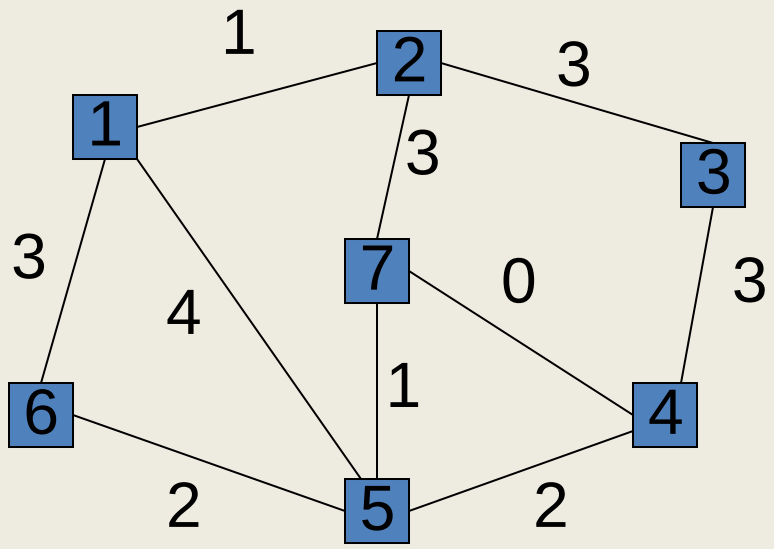
Kruskal's Algorithm for MST

An *edge-based* greedy algorithm

Builds MST by greedily adding edges

1. Initialize with
   - empty MST
   - all vertices marked unconnected
   - all edges unmarked
2. While there are still unmarked edges
   a. Pick the lowest cost edge `(u,v)` and mark it
   b. If `u` and `v` are not already connected, add `(u,v)` to the MST and mark `u` and `v` as connected to each other

Example of for Kruskal



(7,4)	(2,1)	(7,5)	(5,6)	(5,4)	(1,6)	(2,7)	(2,3)	(3,4)	(1,5)
0	1	1	2	2	3	3	3	3	4

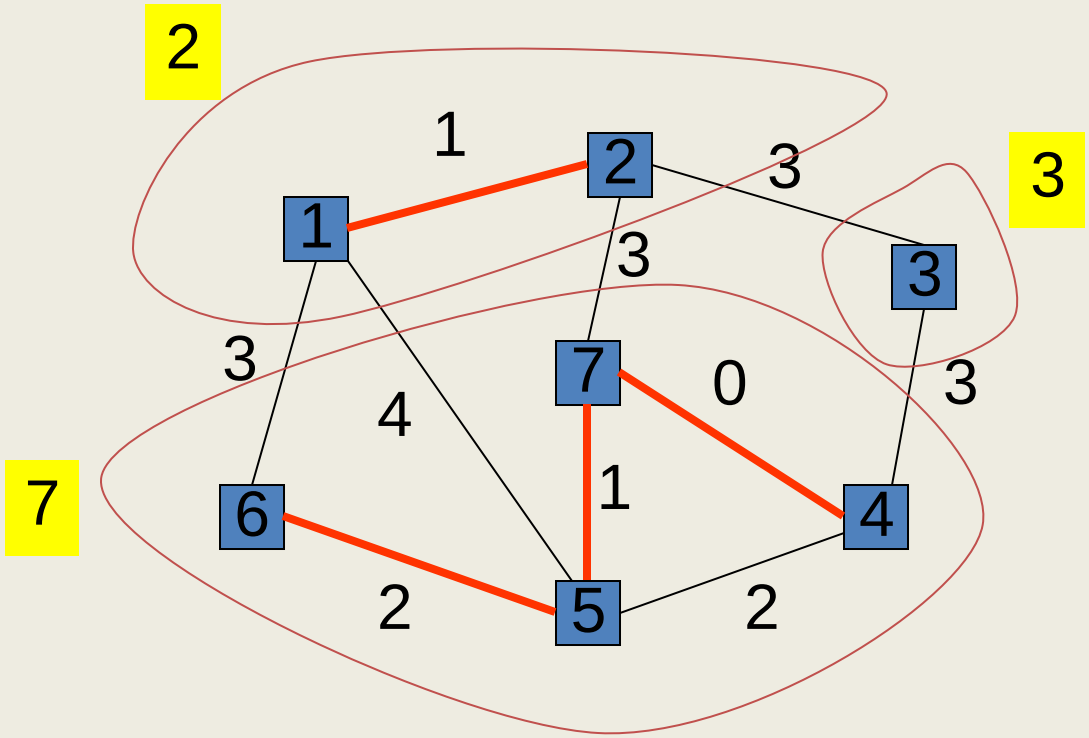
Data Structures for Kruskal

- Sorted edge list

(7,4)	(2,1)	(7,5)	(5,6)	(5,4)	(1,6)	(2,7)	(2,3)	(3,4)	(1,5)
0	1	1	2	2	3	3	3	3	4

- Disjoint Union / Find
  - Union(*a,b*) - merge the disjoint sets named by *a* and *b*
  - Find(a) returns the name of the set containing *a*
- Union / Find data structure will be presented on Friday

Example of DU/F

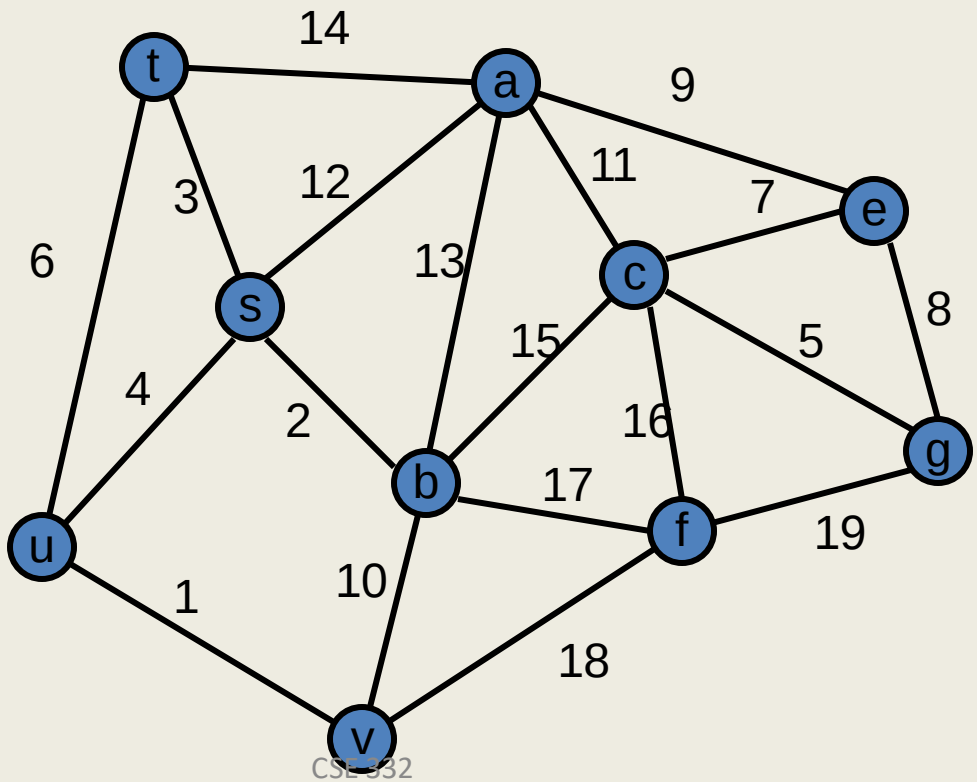


(7,4) (2,1) (7,5) (5,6) (5,4) (1,6) (2,7) (2,3) (3,4) (1,5)

0 1 1 2 2 3 3 3 3 4

Kruskal's Algorithm

- Add the cheapest edge that joins disjoint components



Kruskal's Algorithm with DU / F

```
Sort the edges by increasing cost;
Initialize A to be empty;
for each edge (i,j) chosen in increasing order do
    u := Find(i);
    v := Find(j);
    if not(u = v) then
        add (i,j) to A;
        Union(u,v);
```

This algorithm will work, but it goes through all the edges.

Is this always necessary?

Kruskal code

```
void Graph::kruskal(){
  int edgesAccepted = 0;
  DisjSet s(NUM_VERTICES);

  while (edgesAccepted < NUM_VERTICES - 1){
    e = smallest weight edge not deleted yet;
    // edge e = (u, v)
    uset = s.find(u);
    vset = s.find(v);
    if (uset != vset){
      edgesAccepted++;
      s.unionSets(uset, vset);
    }
  }
}
```

|V| ops to init. sets

|E| heap ops

2|E| finds

|V| unions

Total Cost:

Kruskal's Algorithm: Correctness

It clearly generates a spanning tree. Call it T_K.

Suppose T_K is *not* minimum:

- Pick another spanning tree T_{min} with *lower cost* than T_K
- Pick the smallest edge $e_1=(u,v)$ in T_K that is not in T_{min}
- T_{min} already has a path p in T_{min} from u to v
  - $\Rightarrow$ Adding e_1 to T_{min} will create a cycle in T_{min}
- Pick an edge e_2 in p that Kruskal's algorithm considered *after* adding e_1 (must exist: u and v unconnected when e_1 considered)
  - $\Rightarrow$ cost(e_2) $\geq$ cost(e_1)
  - $\Rightarrow$ can replace e_2 with e_1 in T_{min} without increasing cost!
- Keep doing this until T_{min} is identical to T_K
  - $\Rightarrow$ T_K must also be minimal – contradiction!

Assume the edge costs are distinct

Correctness

Let T_k be the tree found by Kruskal, and let T be a different spanning tree, then T is not a MST

Let e_1 be the minimum cost edge of T_k not in T

If we add e_1 to T, we create a unique cycle A

Let e_2 be the maximum cost edge on A

$c(e_2) > c(e_1)$

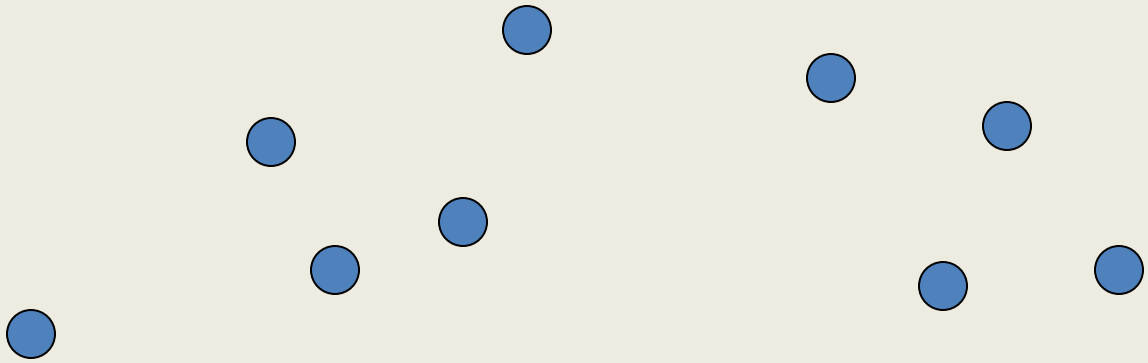
$T' = T + \{e_1\} - \{e_2\}$ is a spanning tree

$C(T') < c(T)$

Therefor, T is not a MST

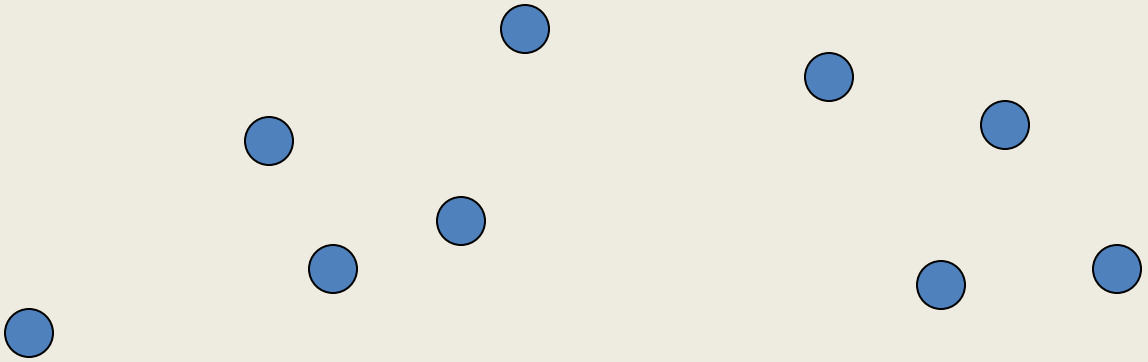
MST Application: Clustering

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

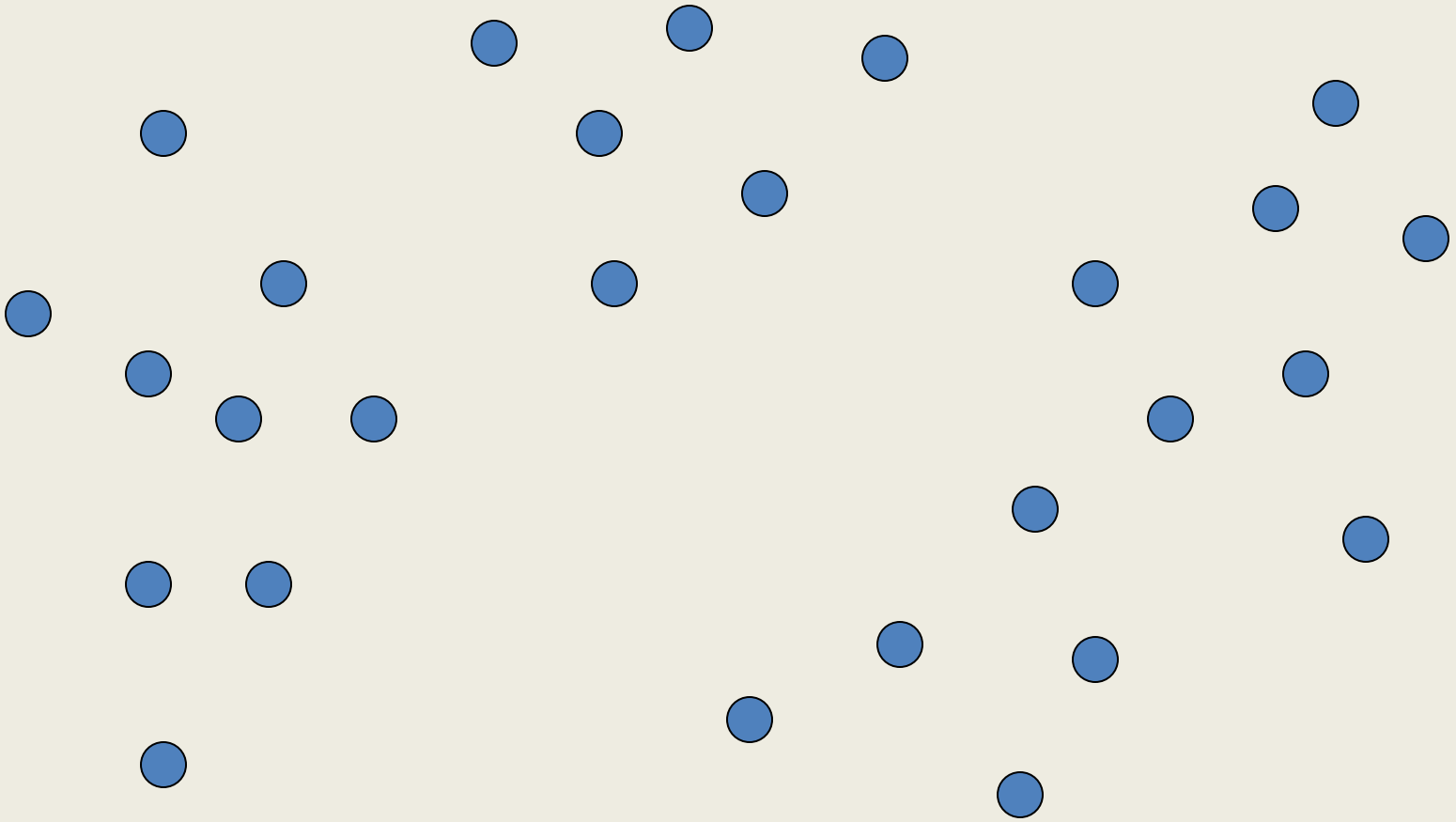


Distance clustering

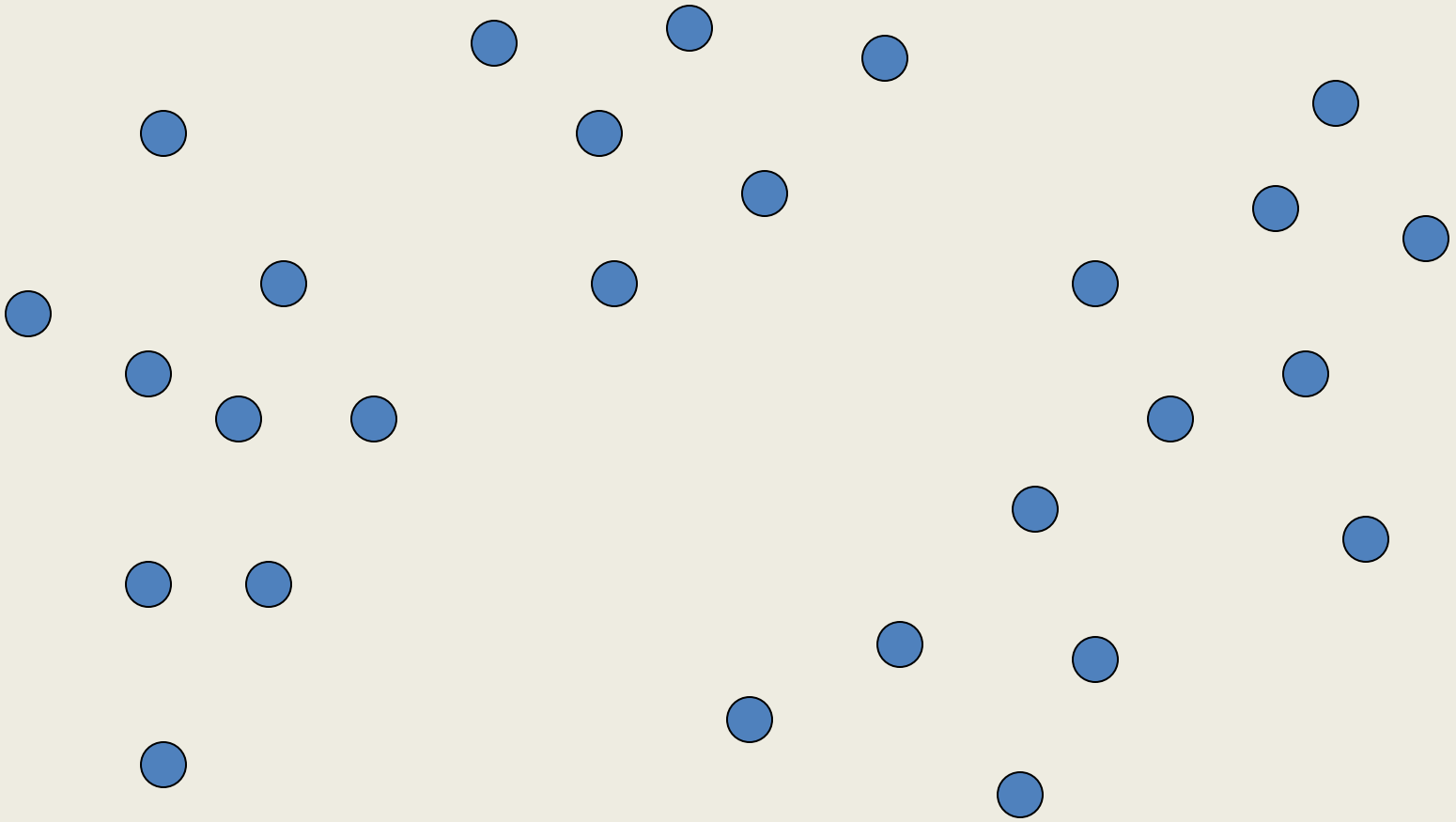
- Divide the data set into K subsets to maximize the distance between any pair of sets
  - dist (S_1, S_2) = min {dist(x, y) | x in S_1, y in S_2}



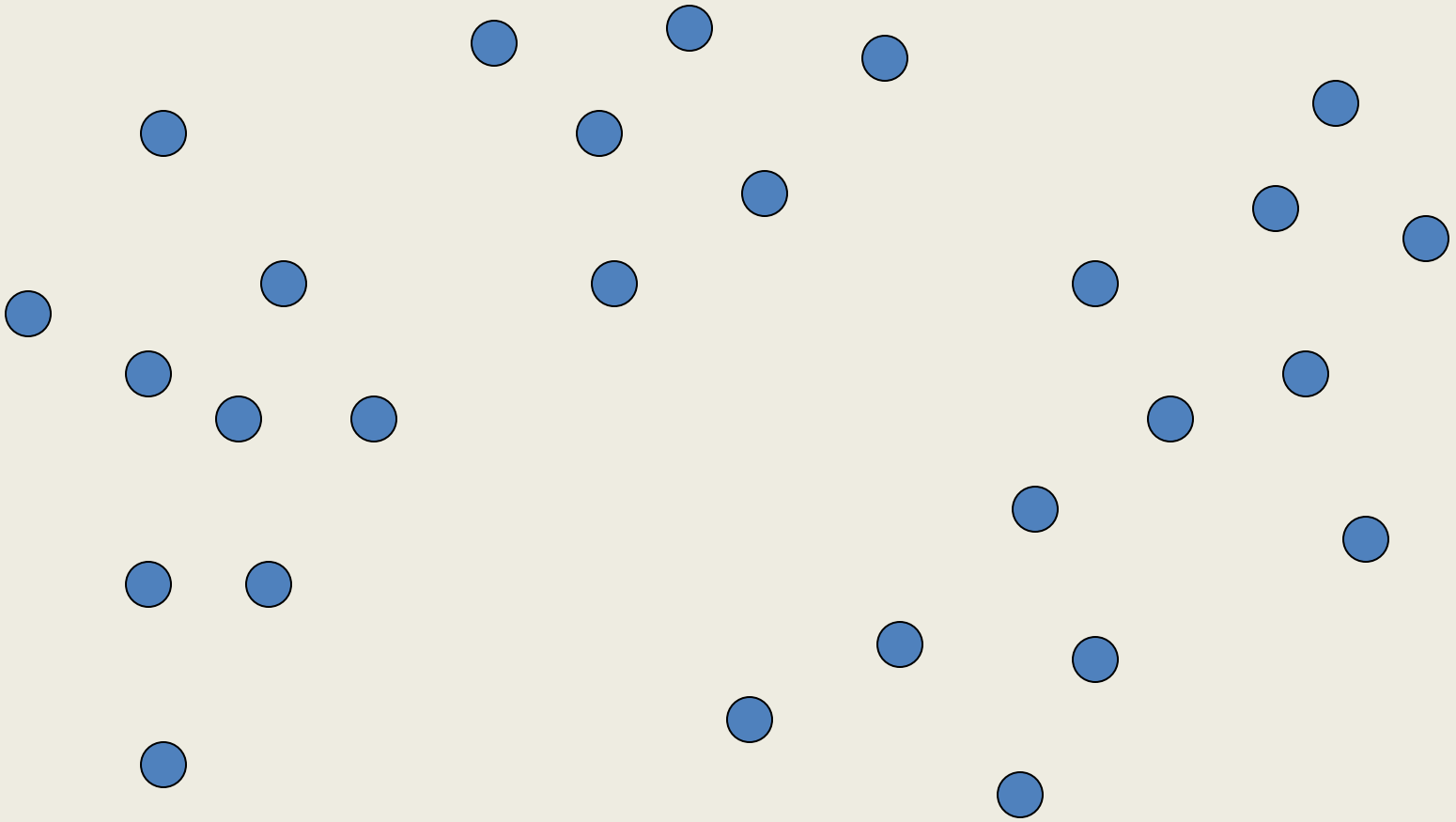
Divide into 2 clusters



Divide into 3 clusters



Divide into 4 clusters



Distance Clustering Algorithm

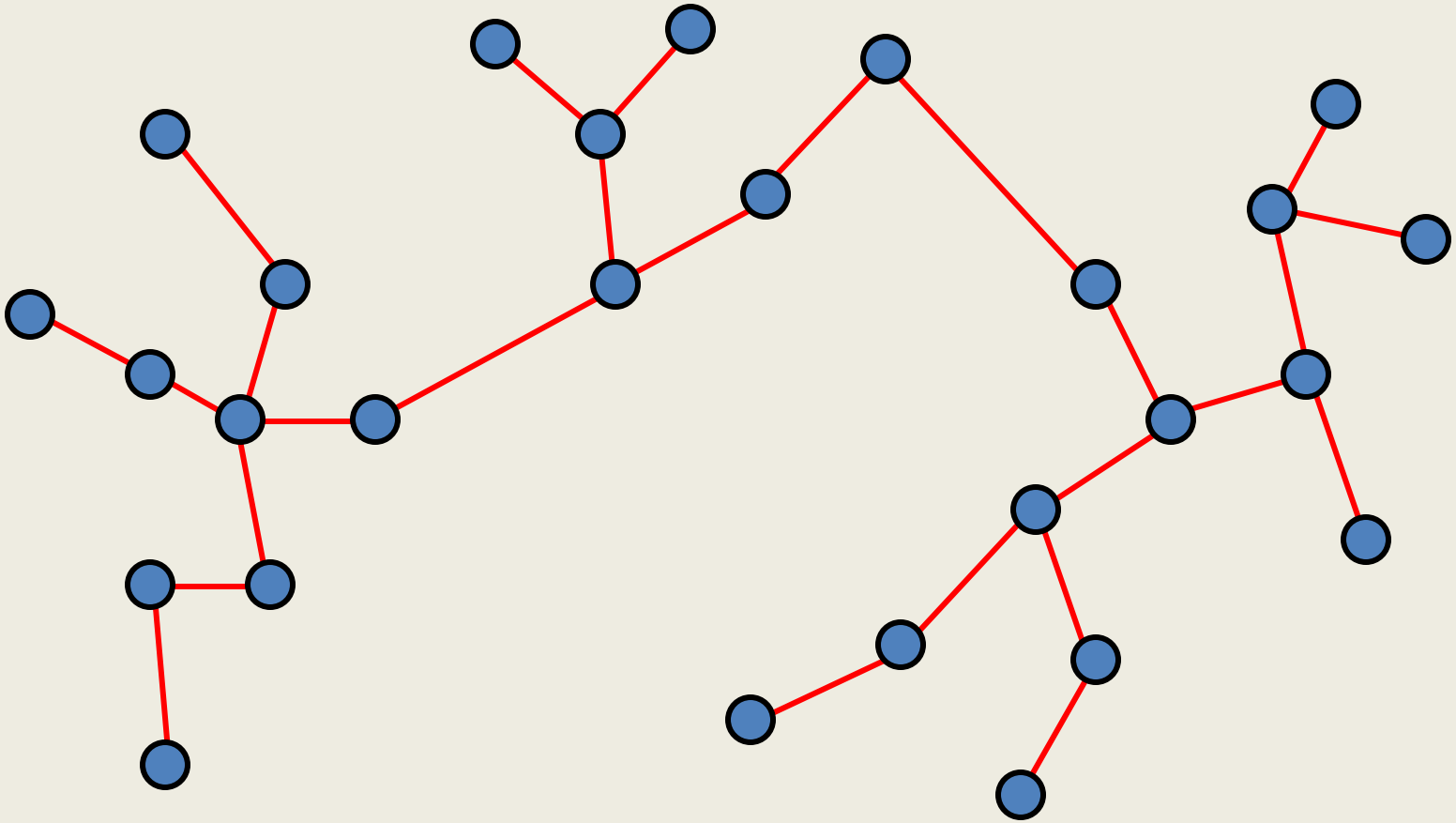
Let C = {{v_1}, {v_2},. . ., {v_n}}; T = { }

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

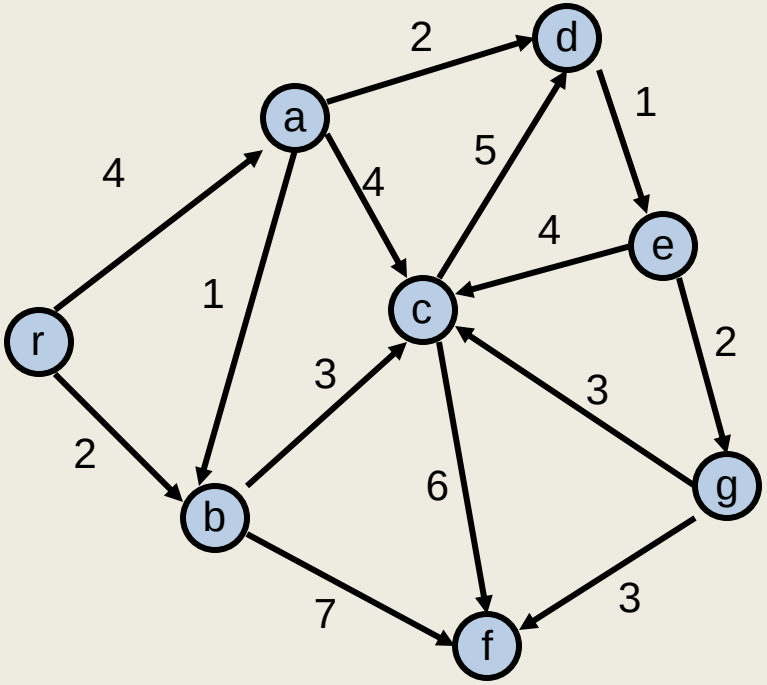
Replace C_i and C_j by C_i U C_j

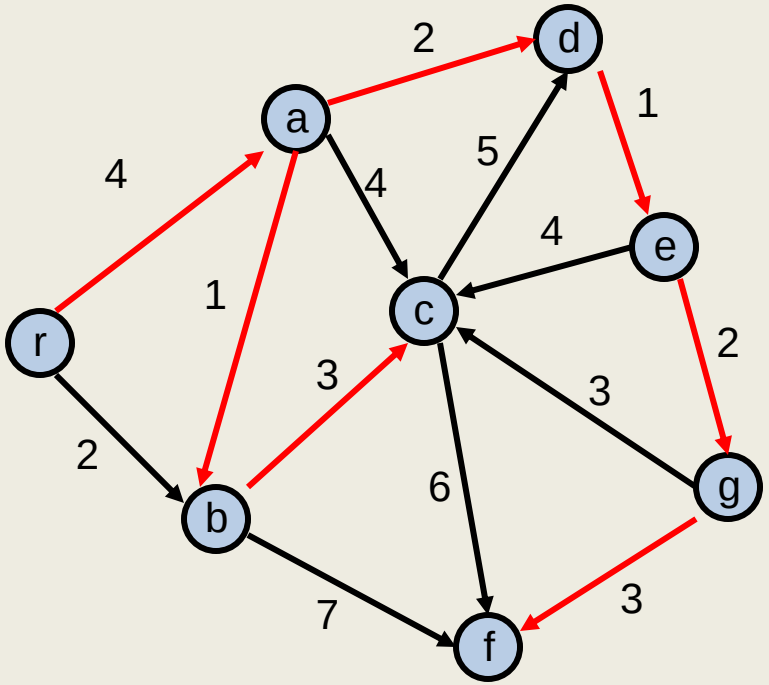
K-clustering



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