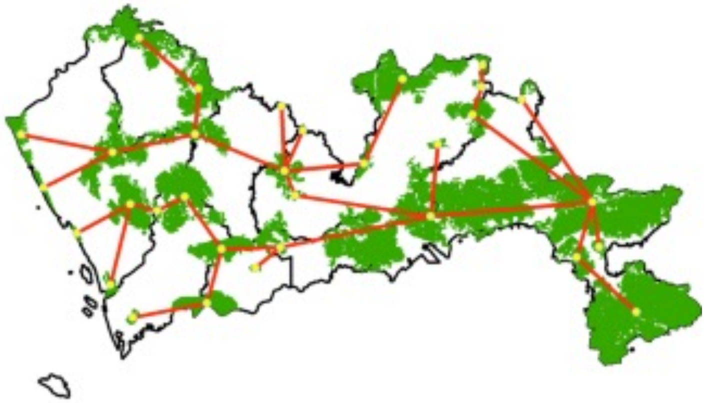
[CSED233-01] Data Structure

Minimum Spanning Trees

Jaesik Park

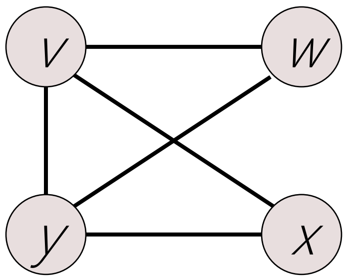
POSTECH



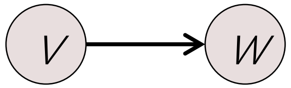
Graph: Terminology

- Graph $G = (V, E)$
  - V: a finite set of vertices/nodes,
    - $n = |V|$: # of vertices
  - E: a finite set of edges/arcs (v, w) where $v, w \in V$
    - $e = |E|$: # of edges

- Example: $G = (V, E)$
  - $V = \{v, w, x, y\}$
  - $W = \{ (v, w), (v, y), (w, y), (y, x), (x, v)\}$



- Two vertices are adjacent if they are connected by an edge
  - v is adjacent to w (w is adjacent from v)
  - (v, w) is incident to w (from v)



Minimum-Cost Spanning Tree

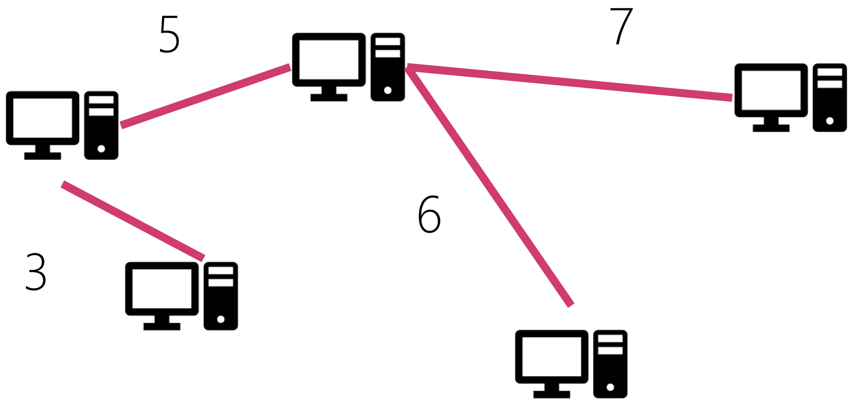
- For a given weighted connected undirected graph G

- Spanning tree T of G
  - A tree that includes all the vertices of the original graph G

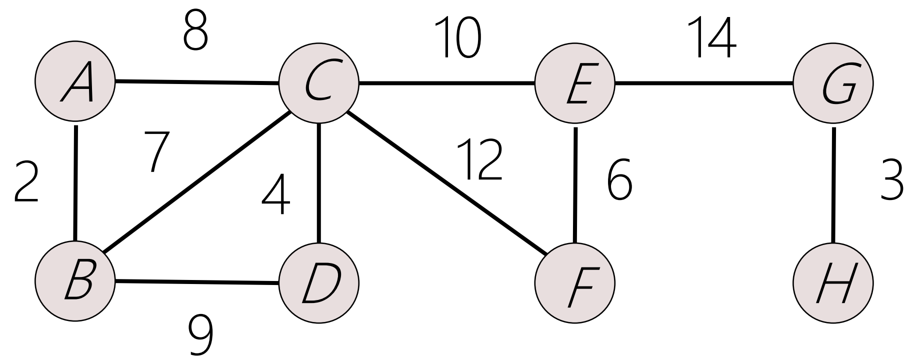
- Minimum-cost spanning tree (MST)
  - A spanning tree whose tree cost is minimum
    - Tree cost is a sum of edge costs

Minimum-Cost Spanning Tree

- Internet
  - Connecting every computers
  - Minimizing cost for network infrastructure
  - Reducing data transferring time
- Road network
  - Connecting every cities
  - Minimizing the total length of highways
- Electronic circuit
- Water pipes



Minimum-Cost Spanning Tree



- A connected graph of 8 vertices with 10 edges
- A spanning tree has only (n – 1) = 7 edges
  - Need to select 7 edges or discard 3 ones

Possible Greedy Strategies (1)

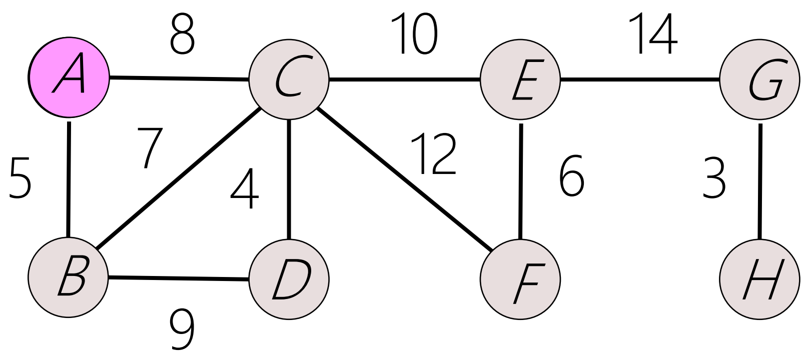
- Prim's algorithm (aka Prim-Jarnik algorithm)
  - Start with a 1-vertex tree and grow it into an n-vertex tree by repeatedly adding a cheapest edge (& a vertex)

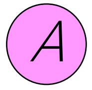
- Kruskal's algorithm
  - Start with an n-vertex forest
  - Consider edges in order of increasing edge cost
  - Select edge if it does not form a cycle together with already selected edges

Possible Greedy Strategies (2)

- Sollin's algorithm
  - Start with an n-vertex forest
  - Each component selects a least-cost edge to connect to another component
  - Eliminate duplicate selections and possible cycles
  - Repeat until only 1 component is left
- Etc.

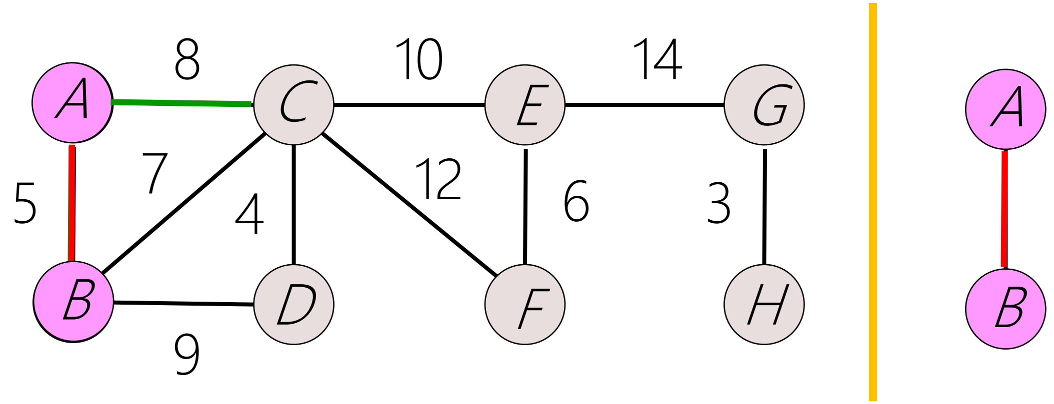
Prim's Algorithm





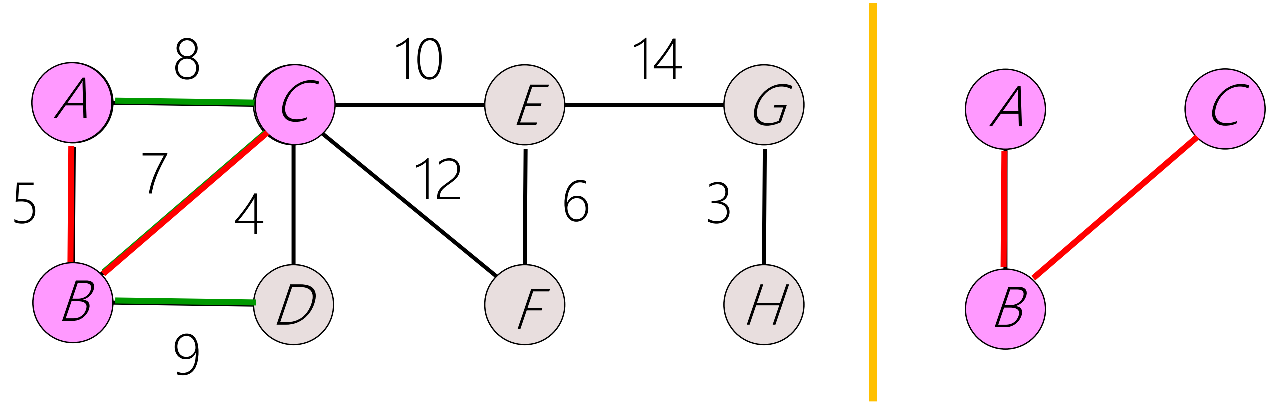
- Start with any single-vertex tree

Prim’s Algorithm



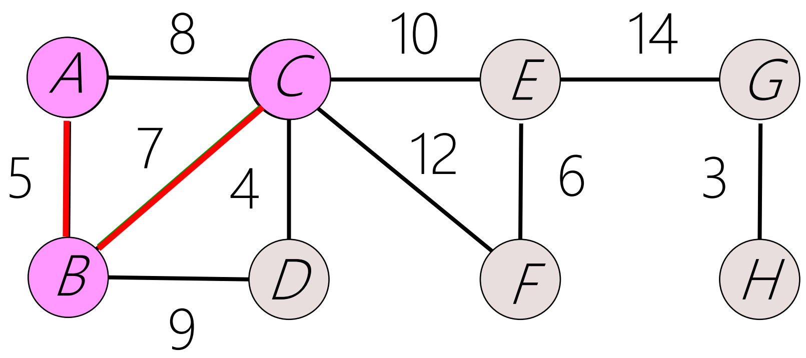
- Start with any single vertex tree
- Get a 2-vertex tree by adding a cheapest edge

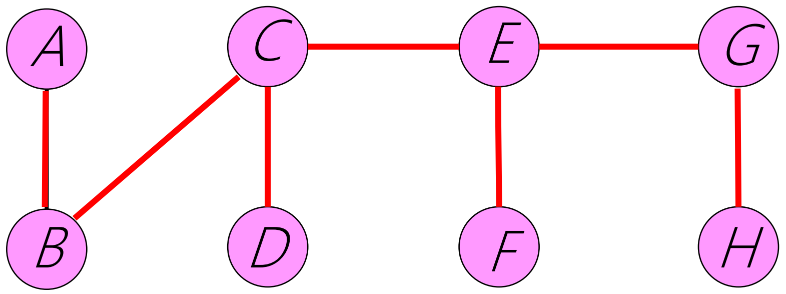
Prim's Algorithm



- Start with any single vertex tree
- Get a 2-vertex tree by adding a cheapest edge
- Get a 3-vertex tree by adding a cheapest edge

Prim's Algorithm





- Start with any single vertex tree
- Get a 2-vertex tree by adding a cheapest edge
- Get a 3-vertex tree by adding a cheapest edge
- Grow the tree one edge at a time until it has $n - 1$ edges (& hence has all n vertices)

Prim's Algorithm: Implementation

- Idea
  - Growing 1-vertex tree into an *n*-vertex tree by repeatedly adding a lowest-cost edge (& its incident vertex)

- Let
  - *Q* : priority queue
    - Contains the vertices NOT contained in the already-generated tree
  - *D[k]* : priority (vertex *k*)
    - the shortest distance (from the "*already-generated*" tree) to vertex *k*
  - *P[k]* : predecessor/parent (vertex *k*)

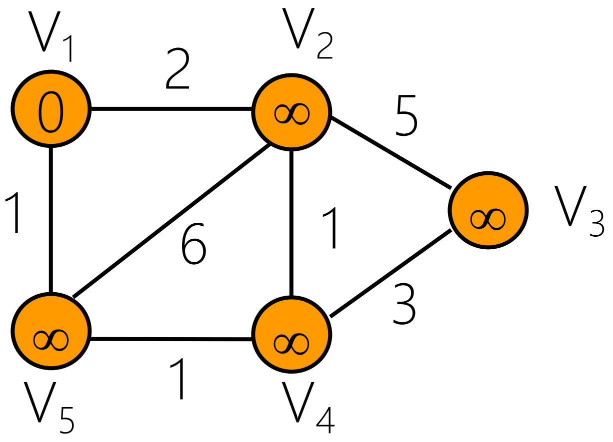
Prim's Algorithm: Implementation

```
Q ← V                                   // initialize Q with all vertices
D[1] := 0; P[1] :=0; and                // initialize priorities D[] & parents P[]
  for all others  D[i] := ∞; P[i] := 0

while  Q not empty  do begin
    w ← DeleteMin(Q)   // w has the lowest D[]
    for  each vertex v ∈ Adj[w]  do
      if  v ∈ Q and C[w, v] < D[v]  then begin
      D[v] := C[w, v];                   // update with shorter cost
      P[v] := w  end;                    // update v's parent as w
end;
```

Prim's Algorithm: Step-by-Step

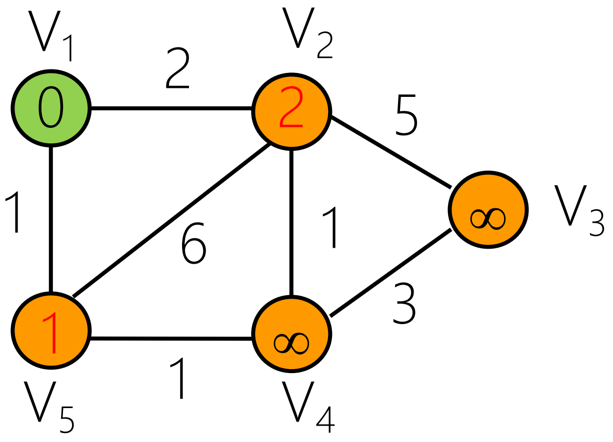
Step-0:



V	in Q	D[i]	P[i]
1	F	0	0
2	F	∞	0
3	F	∞	0
4	F	∞	0
5	F	∞	0

$Q = (V_1, V_2, V_3, V_4, V_5)$

Step-1:

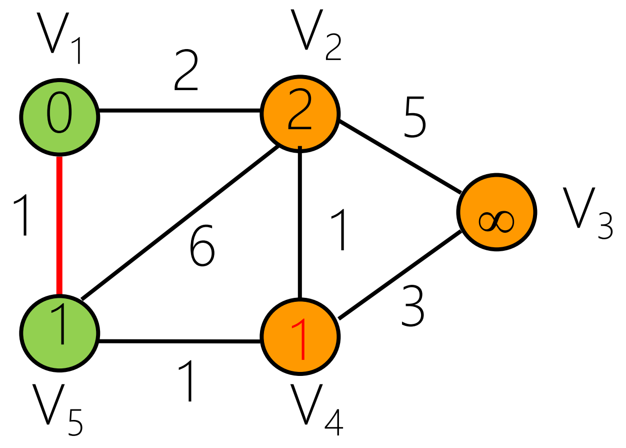


V	in Q	D[i]	P[i]
1	T	0	0
2	F	2	1
3	F	∞	0
4	F	∞	0
5	F	1	1

$Q = (V_5, V_2, V_3, V_4)$

Prim's Algorithm: Step-by-Step

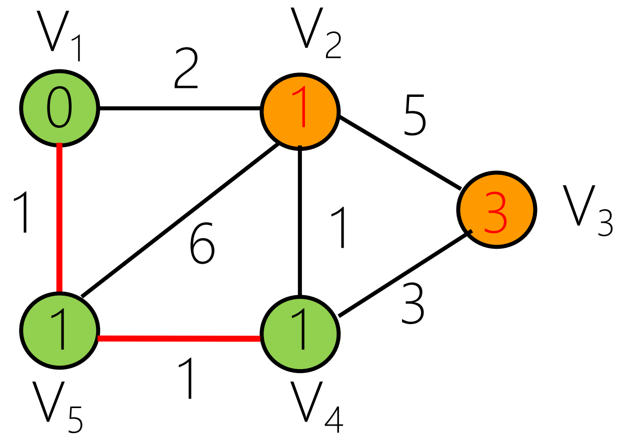
Step-2:



V	in Q	D[i]	P[i]
1	T	0	0
2	F	2	1
3	F	∞	0
4	F	1	5
5	T	1	1

$Q = (V_4, V_2, V_3)$

Step-3:

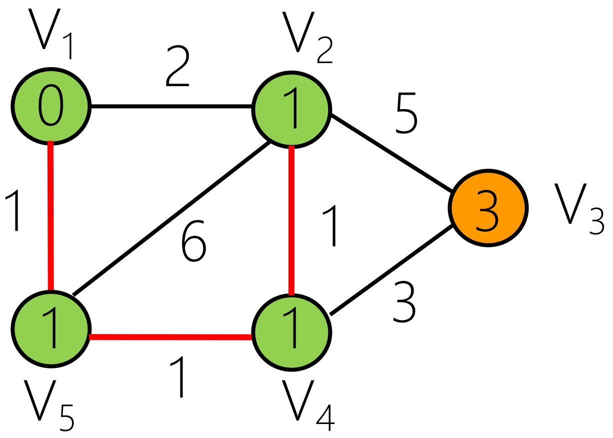


V	in Q	D[i]	P[i]
1	T	0	0
2	F	1	4
3	F	3	4
4	T	1	5
5	T	1	1

$Q = (V_2, V_3)$

Prim's Algorithm: Step-by-Step

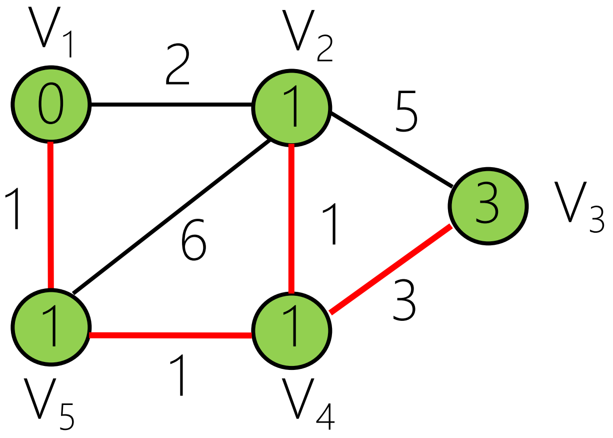
Step-4:



V	in Q	D[i]	P[i]
1	T	0	0
2	T	1	4
3	F	3	4
4	T	1	5
5	T	1	1

$Q = (V_3)$

Step-5:



V	in Q	D[i]	P[i]
1	T	0	0
2	T	1	4
3	T	3	4
4	T	1	5
5	T	1	1

$Q = (\,)$

Why Prim's Algorithm Works (1)

- MST property
  - Let $G = (V, E)$ be a connected graph
    U ($\subset V$): a proper subset of V
    (u, v): an edge of lowest cost s.t. $u \in U$ and $v \in V - U$

    Then, there exists an MST that includes the edge (u, v)

- MST property satisfies the greedy-choice property
  - A globally-optimal solution can be arrived at by making a locally-optimal choice

Why Prim's Algorithm Works (2)

- How to prove the MST property?

- Proof by contradiction
  - Assume that there is no MST including (u, v)
    - By adding (u, v) to any MST T, which does not include (u, v), we create a cycle
    - That is, there must exist an edge (u', v') s.t. $u' \in U$ and $v' \in V-U$
  - $c(u, v) \leq c(u', v')$ by assumption
  - By replacing (u', v') by (u, v), we can get another MST which contains (u, v)
  - Contradiction!

Complexity: Prim's Algorithm

- Depending on the choice of data structure
  - Adjacent matrix & searching
    - Simple implementation
    - $O(n^2)$ – better for dense graph
  - Adjacent list & min-heap
    - $O(e \log n)$ – better for sparse graph

Prim's: Adjacent Matrix (Array)

$O(n)$

```
Initialize Q ← V
D[1] := 0; P[1] :=0; and
  for all other D[i] := ∞; P[i] := 0

while  Q not empty  do begin
  w ← DeleteMin(Q)
  for  each vertex v ∈ Adj[w]  do
    if  v ∈ Q and C[w, v] < D[v] then begin
      D[v] := C[w, v];
      P[v] := w  end;
end;
```

n times

$O(n)$

deg(w) times

$O(1)$

➔ Time complexity = ? $O(n) + O(n*n + e*1) = O(n^2)$

Prim's: Adjacent List & Min-Heap

$O(n)$

```
Initialize Q ← V
D[1] := 0; P[1] :=0; and
  for all other D[i] := ∞; P[i] := 0

while  Q not empty  do begin
  w ← DeleteMin(Q)
  for  each vertex v ∈ Adj[w]  do
    if  v ∈ Q and C[w, v] < D[v] then begin
      D[v] := C[w, v];
      P[v] := w  end;
end;
```

$O(\log n)$

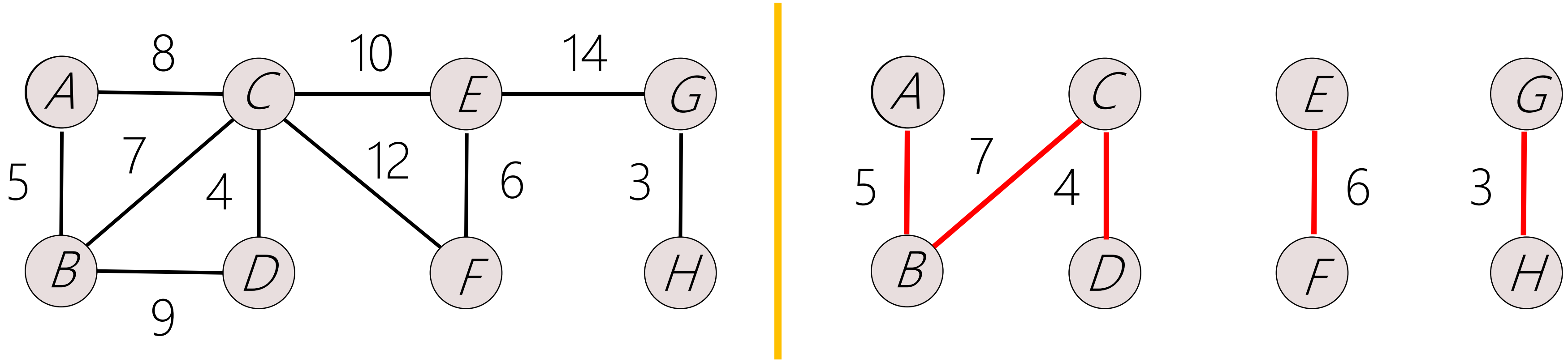
n times

deg(w) times

$O(\log n)$

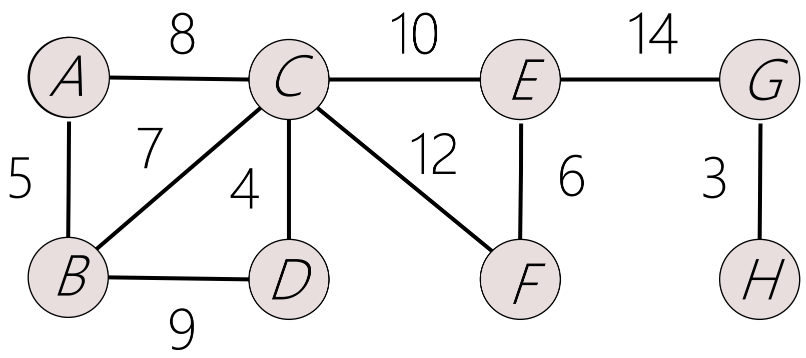
➔ Time complexity = ? $O(n) + O(n \log n + e \log n) = O(e \log n)$

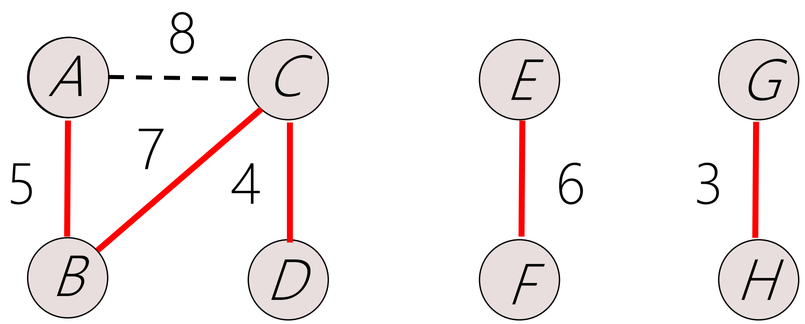
Kruskal's Algorithm



- Start with a forest that has no edge
- Consider edges in ascending order of edge cost
  - Edge (G, H) is considered first and added to the forest
  - Edges (C, D), (A, B), (E, F), (B, C) are considered in sequence & added

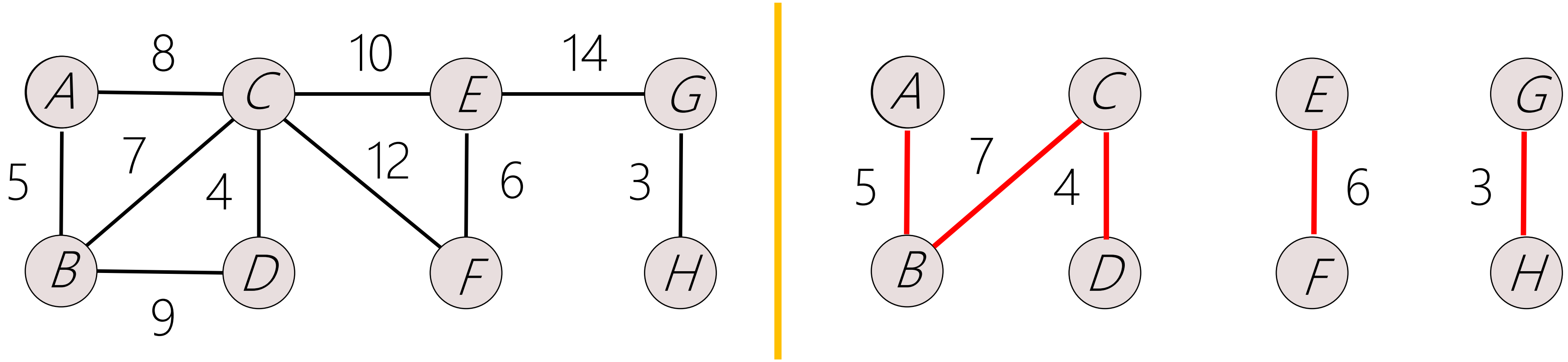
Kruskal's Algorithm





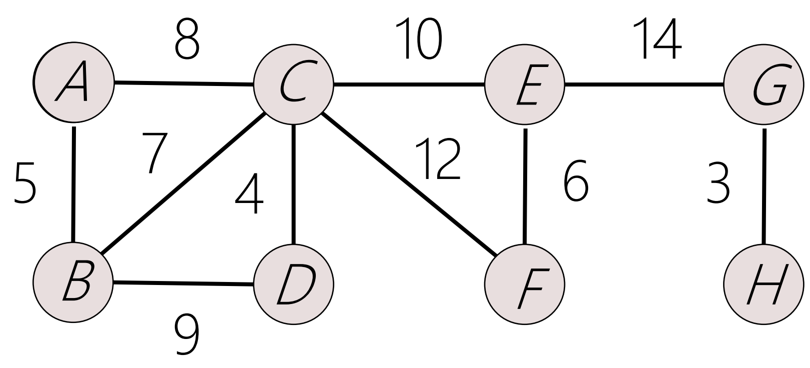
- Start with a forest that has no edge
- Consider edges in ascending order of edge cost
  - Edge (G, H) is considered first and added to the forest
  - Edges (C, D), (A, B), (E, F), (B, C) are considered in sequence & added
  - Edge (A, C) is considered next

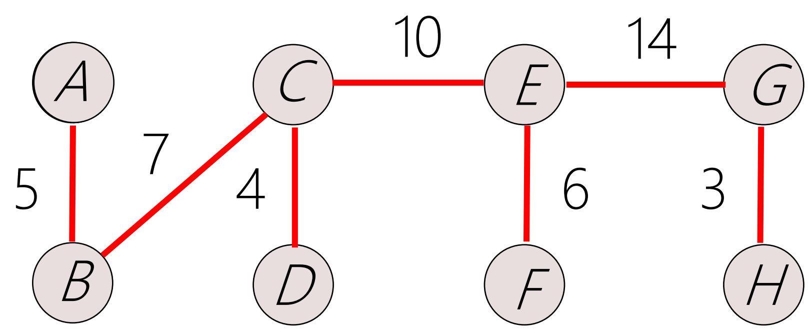
Kruskal's Algorithm



- Start with a forest that has no edge
- Consider edges in ascending order of edge cost
  - Edge (G, H) is considered first & added to the forest
  - Edges (C, D), (A, B), (E, F), (B, C) are considered in sequence & added
  - Edge (A, C) is considered next
    - But it is rejected because it creates a cycle

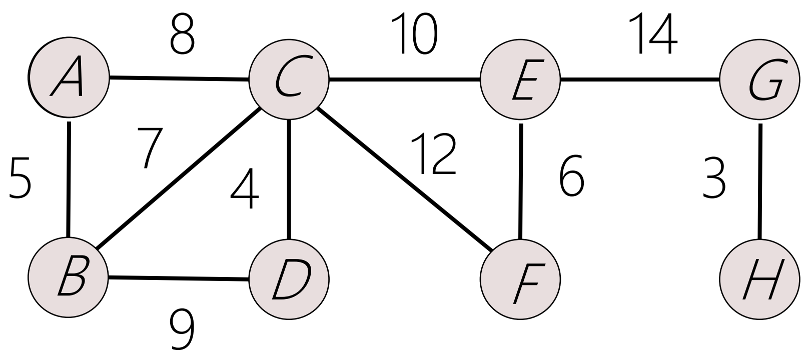
Kruskal's Algorithm

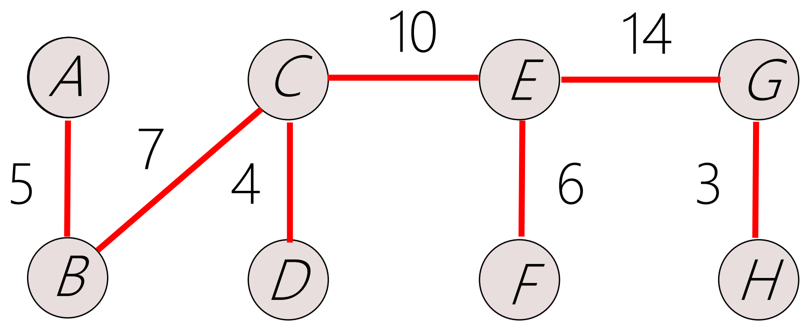




- Consider edges in ascending order of edge cost
  - Edge (B, D) is considered next, but rejected because it creates a cycle
  - Edge (C, E) is considered next, & added
  - Edge (C, F) is considered next, but rejected because it creates a cycle
  - Edge (E, G) is considered next, & added

Kruskal's Algorithm





- A total of $(n - 1)$ edges are selected with no cycle formed
- So, we must have an MST whose cost is 49

- Are there any other MST?
  - MST is unique when all edge costs are different

Complexity: Kruskal's Algorithm

- Kruskal's algorithm (recap)
  - Sort edges by cost & examine them from the cheapest
  - Put each edge into the current forest if it doesn't form a cycle
    - To do this efficiently, we need a data structure that can support Union-Find operations

Complexity: Kruskal’s Algorithm

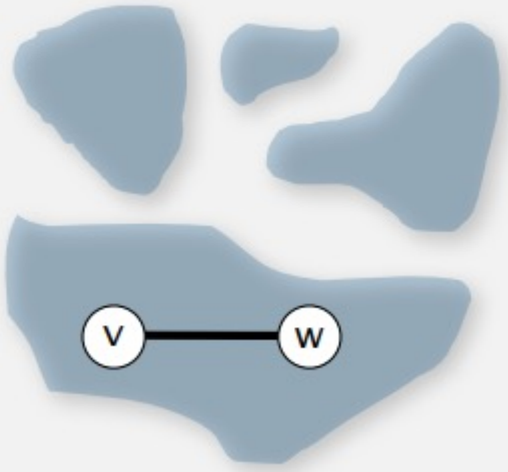
- Union-Find problem

  - Put each edge into the current forest if it doesn’t form a cycle

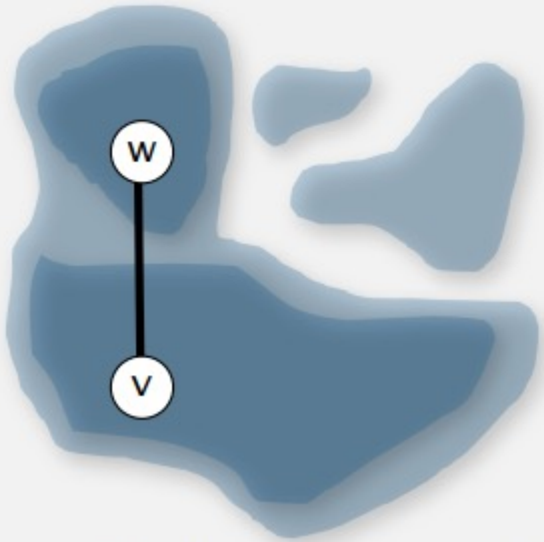
$$\Updownarrow$$

  - For each edge,
    - If it connects two different components (FIND), insert the edge, merging the two components (UNION)
    - Otherwise, that is, if the two nodes are in the same component (FIND), then we will skip this edge

Union Find



Case 1: adding v–w creates a cycle



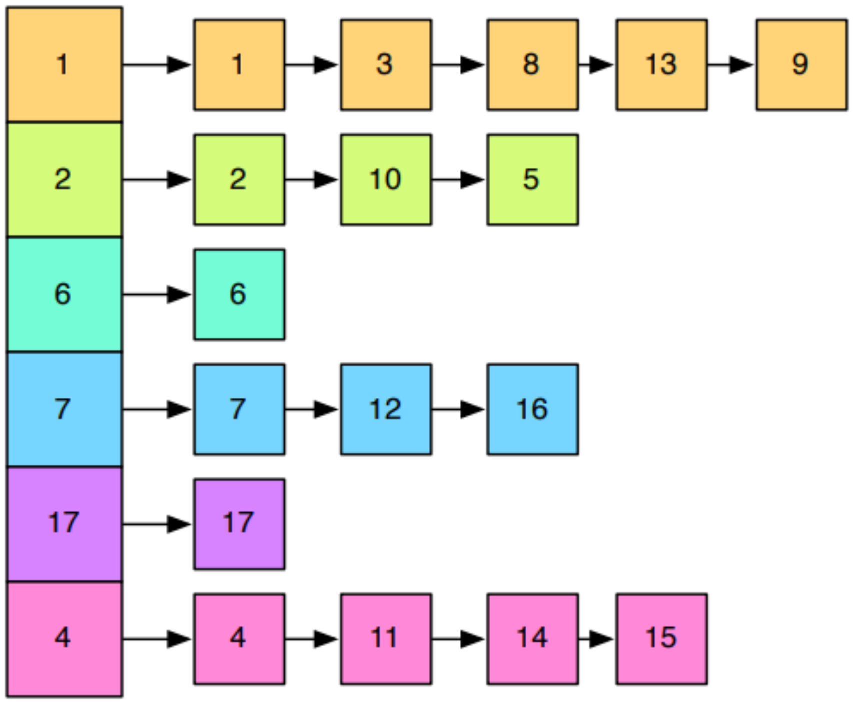
Case 2: add v–w to T and merge sets containing v and w

Union-Find Abstract Data Type

- UF.create(S)
  - create the data structure containing |S| sets, each containing one item from S.
- UF.find(i)
  - return the “name” of the set containing item i.
- UF.union(a,b)
  - merge the sets with names a and b into a single set.

A Union-Find Data Structure

UF Items:



UF Sizes:

1	5
2	3
6	1
7	3
17	1
4	4

UF Sets Array:

1	2	1	4	2	6	7	1	1	2	4	7	1	4	4	7	17
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17

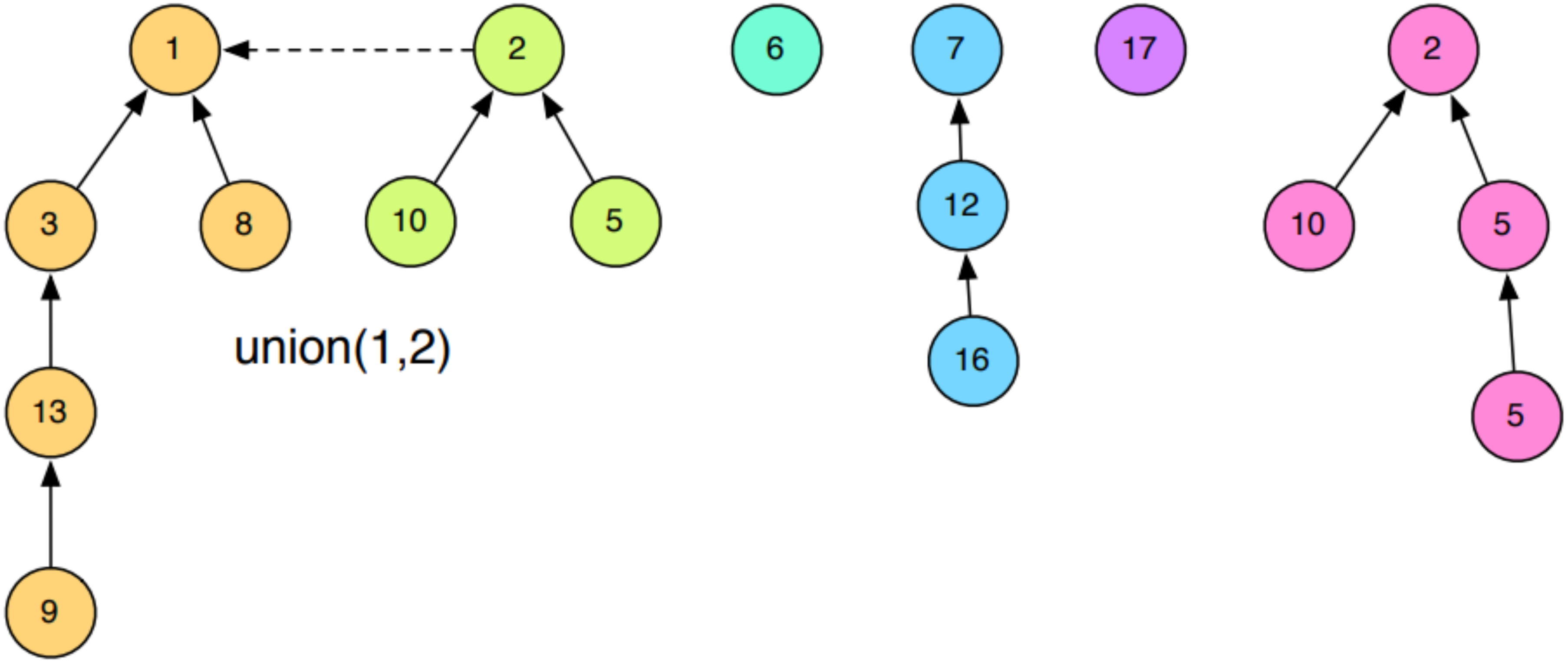
Implementing the union & find operations

- make_union_find(S)
  - Create data structures on previous slide.
  - Takes time proportional to the size of S.

- find(i)
  - Return UF.sets[i].
  - Takes a constant amount of time.

- union(x,y)
  - Use the "size" array to decide which set is smaller.
  - Assume x is smaller.
  - Walk down elements i in set x, setting sets[i] = y.
  - Set size[y] = size[y] + size[x].

Another way to implement Union-Find



Tree-based Union-Find

- make union find(S)
  - Create |S| trees each containing a single item and size 1.
  - Takes time proportional to the size of S.

- find(i)
  - Follow the pointer from i to the root of its tree.

- union(x,y)
  - If the size of set x is < that of y, make y point to x.
  - Takes constant time.

Complexity: Kruskal's Algorithm

- Kruskal's algorithm (using min-heap)
  - Sort edges by cost (i.e., construct a min-heap)
    - O(e log e) or
    - O(e) (if we heapify them all at once)
  - Repeat up to the-number-of-edges times
    - Find the least-cost edge from min-heap
      - O(log e)
    - Does it form a cycle (FIND)? If not, put it into the forest (UNION)
      - O(log e) for *union/find* operations

➔ Time complexity = ? O(e log e) = O(e log n)

- Prim's method is faster for dense graph, but Kruskal's for sparse case

Possible Greedy Strategies (1)

- Prim's algorithm (aka Prim-Jarnik algorithm)
  - Start with a 1-vertex tree and grow it into an *n*-vertex tree by repeatedly adding a cheapest edge (& a vertex)

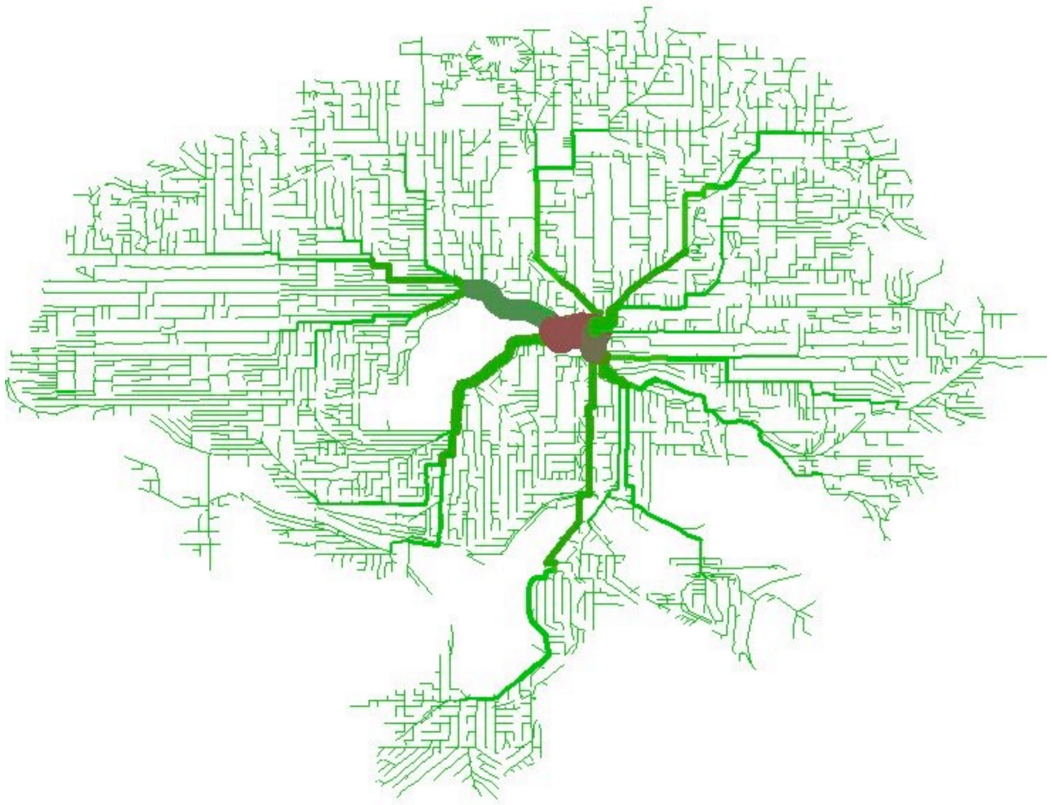
- Kruskal's algorithm
  - Start with an *n*-vertex forest
  - Consider edges in order of increasing edge cost
  - Select edge if it does not form a cycle together with already selected edges

Possible Greedy Strategies (2)

- Sollin's algorithm
  - Start with an n-vertex forest
  - Each component selects a least-cost edge to connect to another component
  - Eliminate duplicate selections and possible cycles
  - Repeat until only 1 component is left

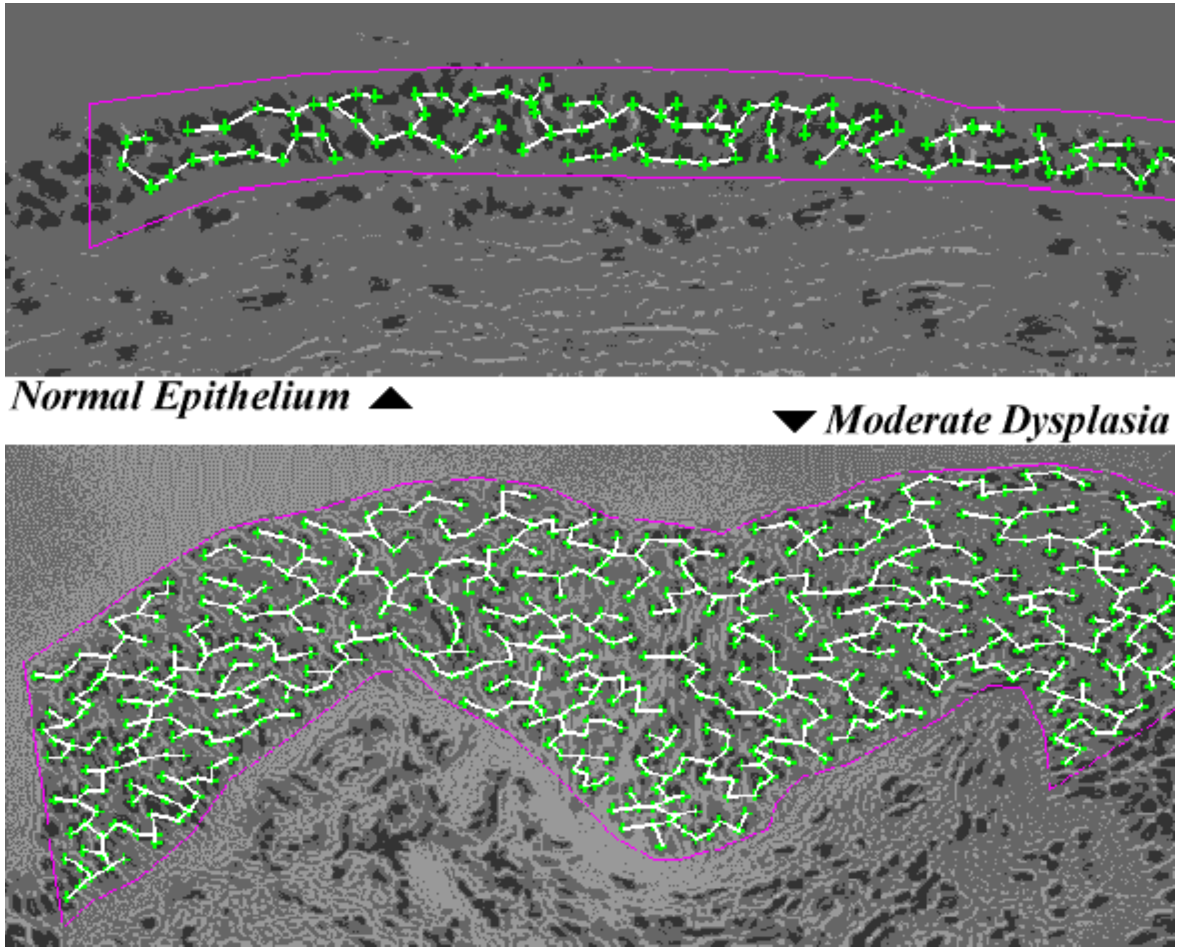
- Etc.

MST of bicycle routes in North Seattle



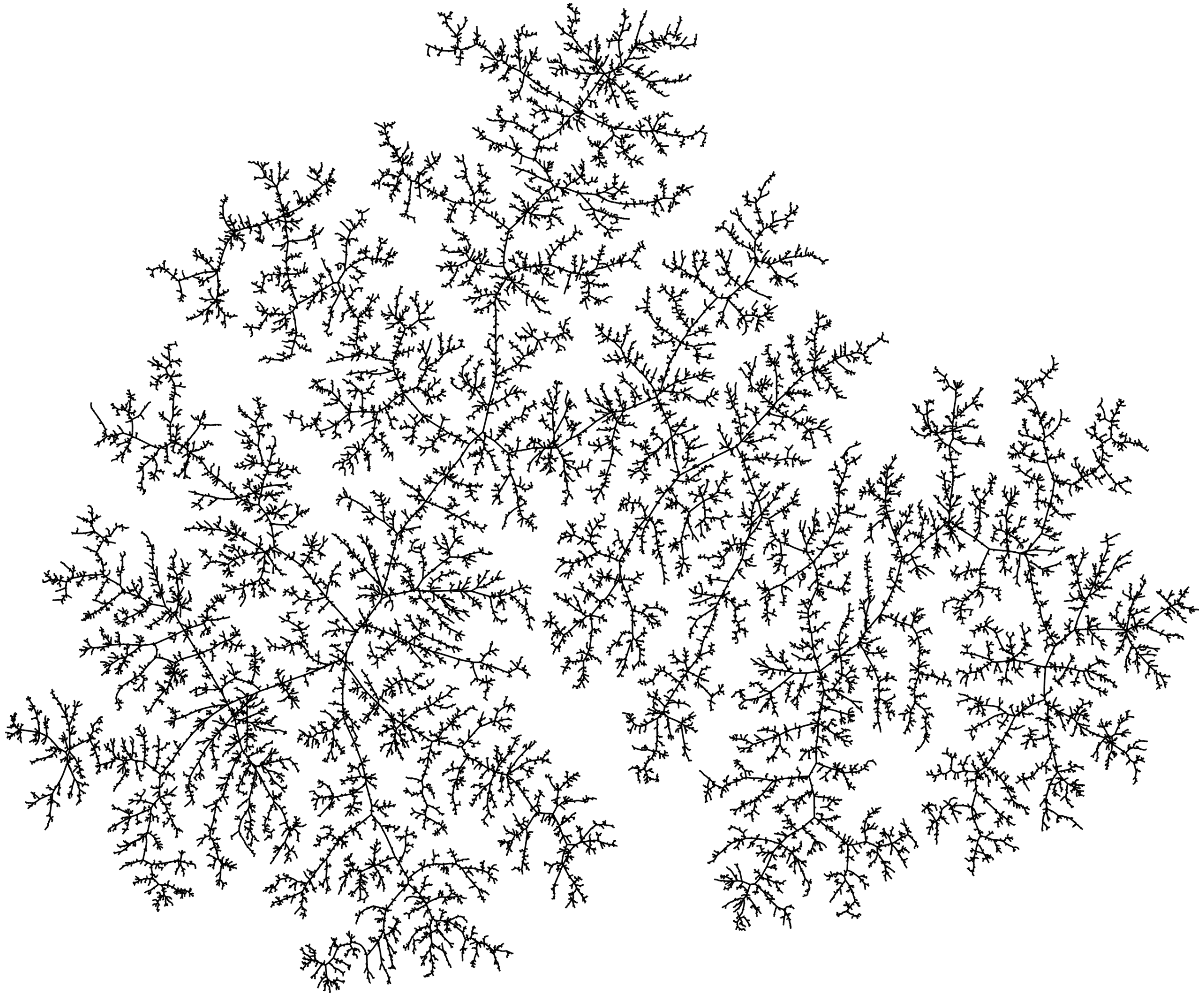
http://www.flickr.com/photos/ewedistrict/21980840

MST describes arrangement of nuclei in the epithelium for cancer research



http://www.bccrc.ca/ci/ta01_archlevel.html

MST of random graph



http://algo.inria.fr/broutin/gallery.html

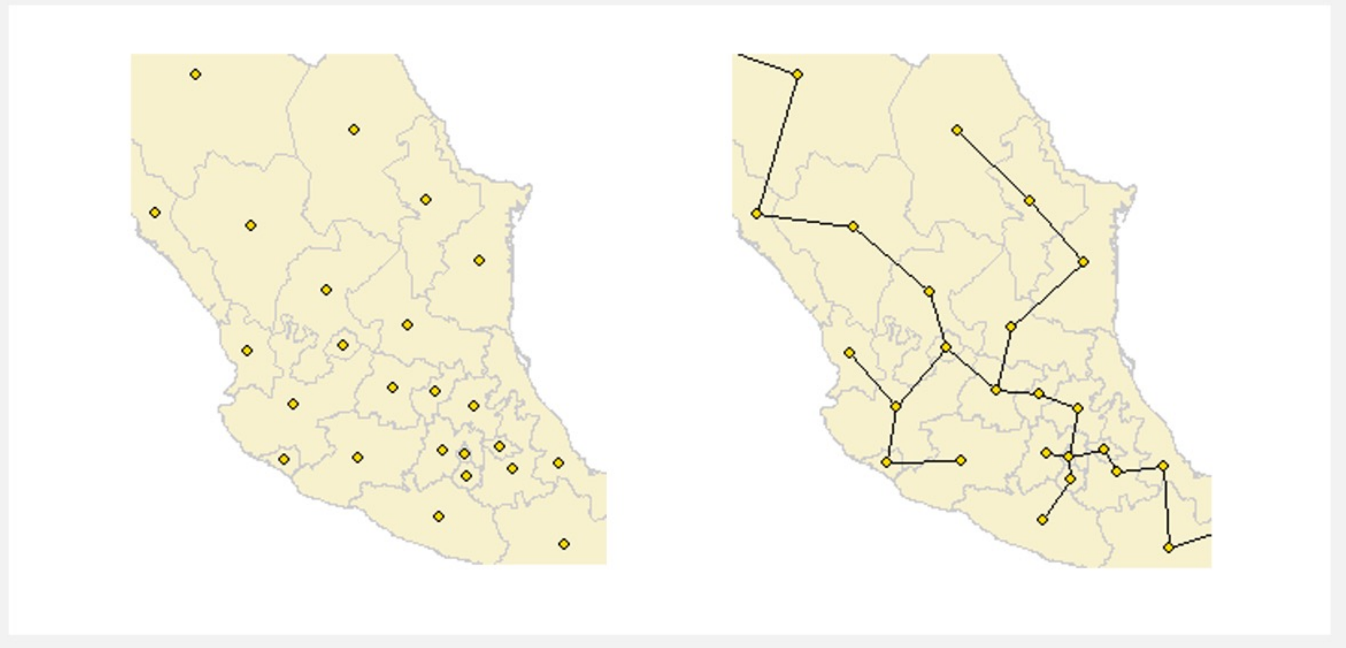
Does a linear-time MST algorithm exist?

deterministic compare-based MST algorithms

year	worst case	discovered by
1975	$E \log \log V$	Yao
1976	$E \log \log V$	Cheriton-Tarjan
1984	$E \log^* V,\ E + V \log V$	Fredman-Tarjan
1986	$E \log (\log^* V)$	Gabow-Galil-Spencer-Tarjan
1997	$E\ \alpha(V) \log \alpha(V)$	Chazelle
2000	$E\ \alpha(V)$	Chazelle
2002	*optimal*	Pettie-Ramachandran
20xx	E	???

Euclidean MST

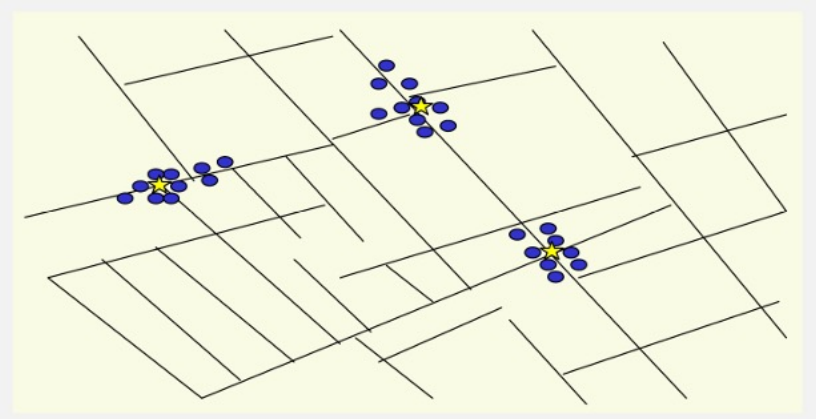
- Given N points in the plane, find MST connecting them, where the distances between point pairs are their Euclidean distances.



- Brute force. Compute ~ N^2 / 2 distances and run Prim's algorithm.
- Ingenuity. Exploit geometry and do it in ~ c N log N.

Scientific application: clustering

- k-clustering
  - Divide a set of objects classify into k coherent groups
- Distance function
  - Numeric value specifying "closeness" of two objects.
- Goal
  - Divide into clusters so that objects in different clusters are far apart.

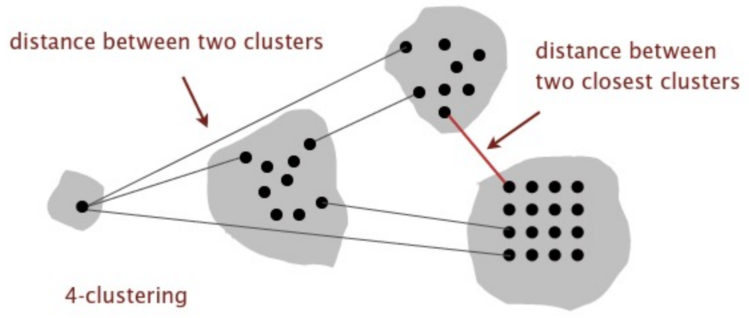


outbreak of cholera deaths
in London in 1850s
(Nina Mishra)

- Applications.
  - Routing in mobile ad hoc networks.
  - Document categorization for web search.
  - Similarity searching in medical image databases.
  - Skycat: cluster sky objects into stars, quasars, galaxies.

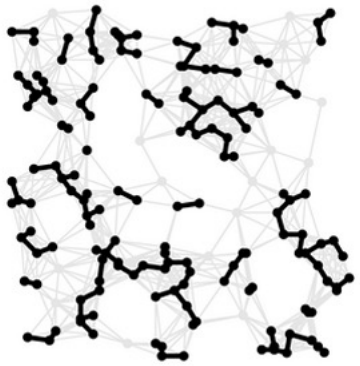
Single-link clustering

- Single link
  - Distance between two clusters equals the distance between the two closest objects (one in each cluster).
- Single-link clustering
  - Given an integer k, find a k-clustering that maximizes the distance between two closest clusters.



Single-link clustering algorithm

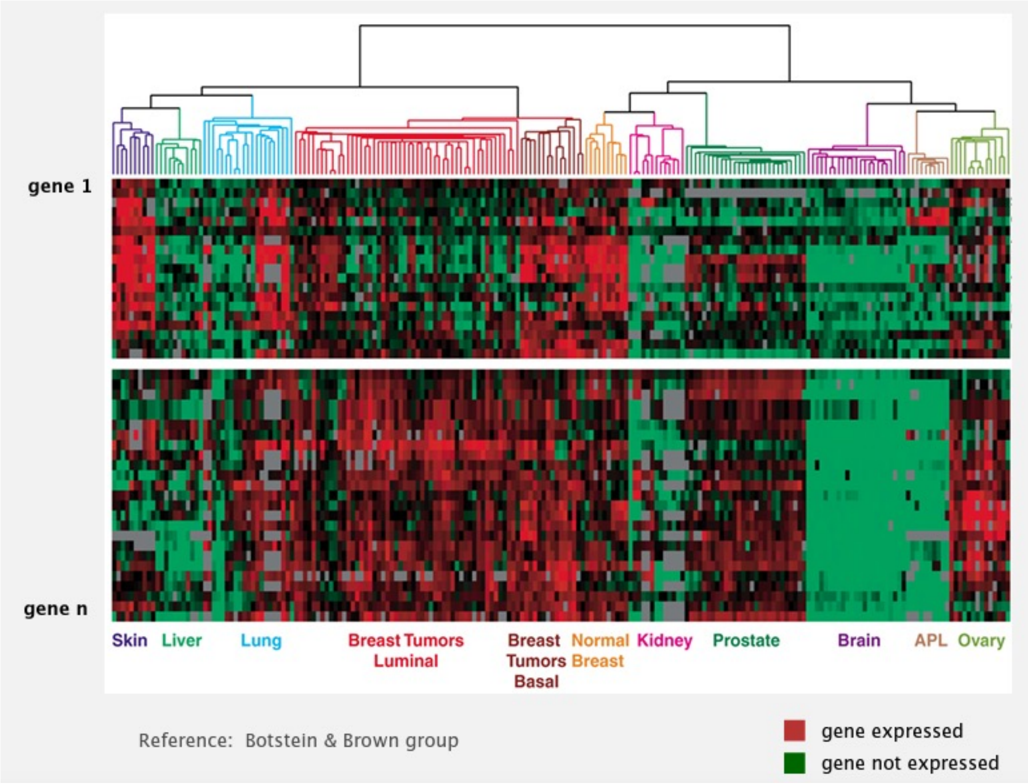
- "Well-known" algorithm in science literature for single-link clustering
  - Form V clusters of one object each
  - Find the closest pair of objects such that each object is in a different cluster, and merge the two clusters
  - Repeat until there are exactly k clusters.



- Observation
  - This is Kruskal's algorithm. (stopping when k connected components)

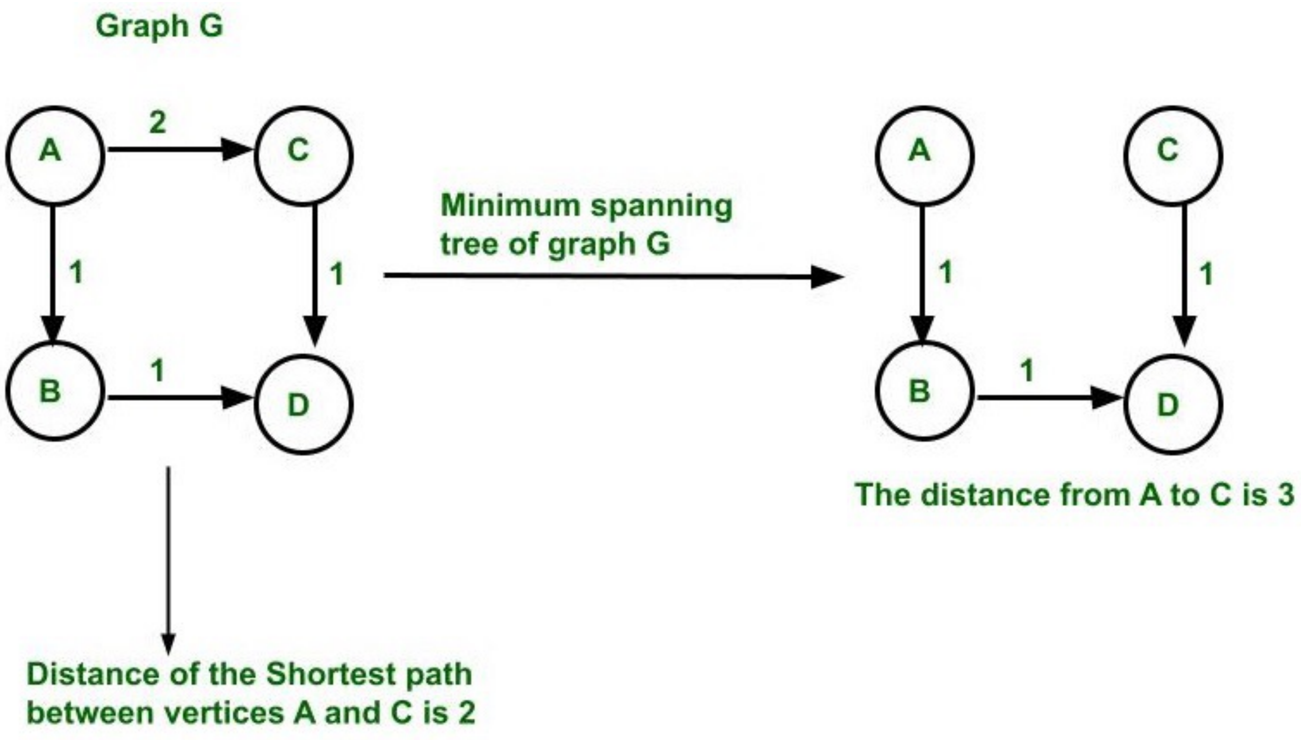
Dendrogram of cancers in human

- Tumors in similar tissues cluster together.



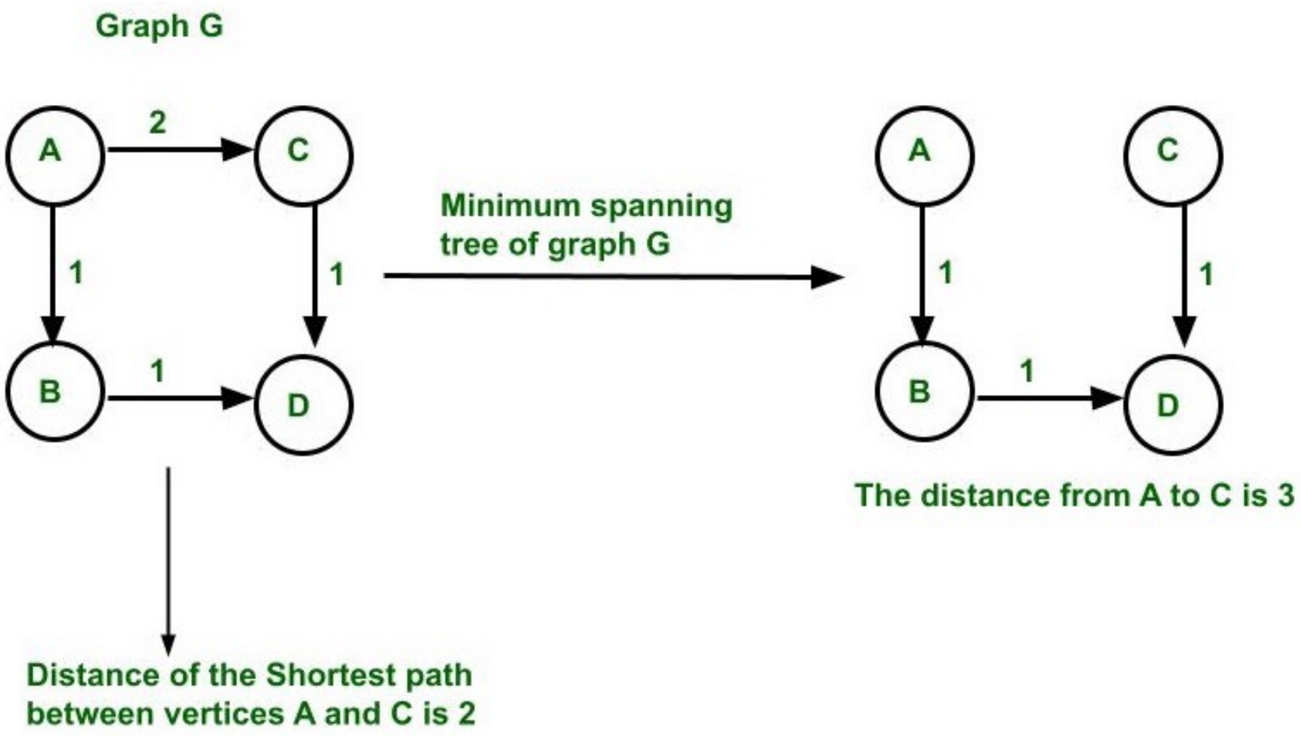
Minimum Spanning Tree and Shortest Path

- They are not the same thing



Uniqueness of MST

- If each edge has a distinct weight then there will be only one



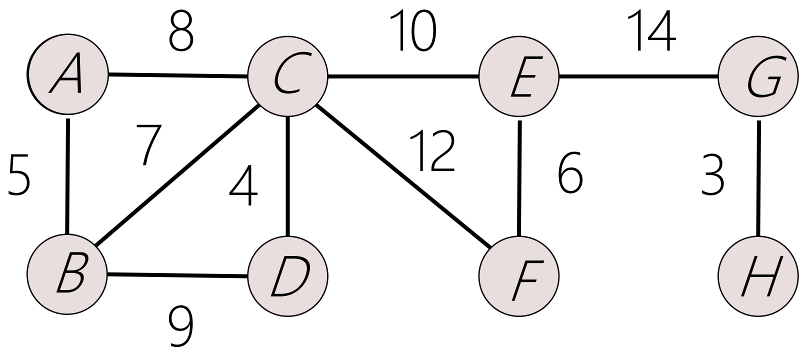
Sollin’s Algorithm

- First published in 1926 by Otakar Borůvka
  - constructing an efficient electricity network for Moravia.
- Frequently called as Sollin's algorithm or Borůvka’s algorithm

- Method
  - Start with an *n*-vertex forest
  - Each component selects a least-cost edge to connect to another component
  - Eliminate duplicate selections and possible cycles
  - Repeat until only 1 component is left

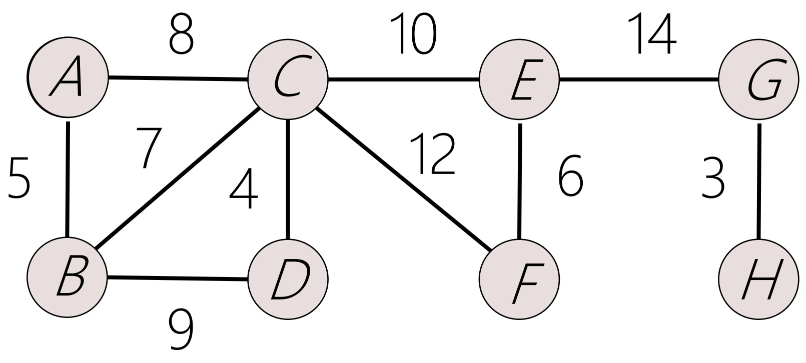
- Greedy?

Sollin's Algorithm



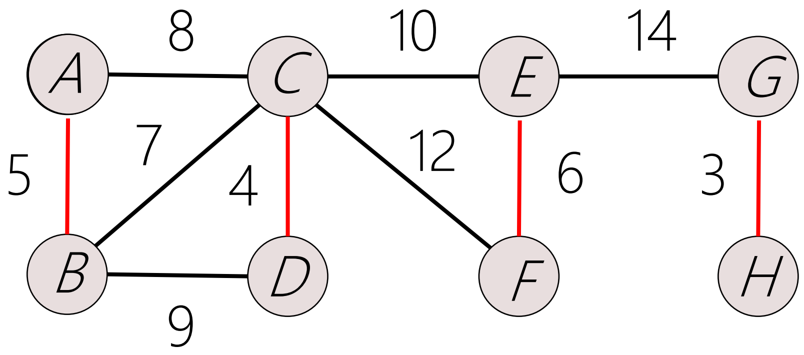
- Start with a forest that has no edge

Sollin's Algorithm



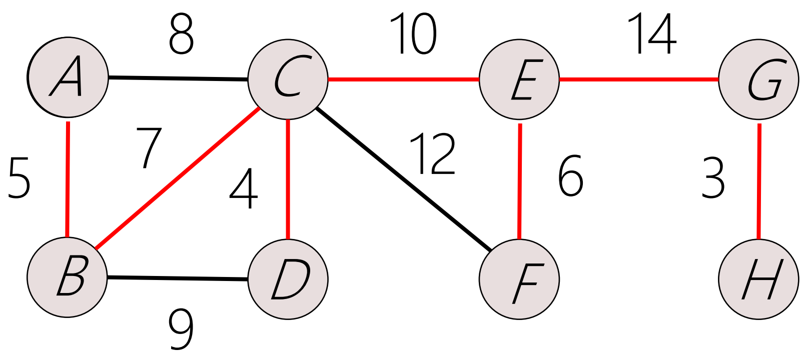
- For each cluster, select the minimum cost inter-cluster edge
  - A: 5
  - B: 5
  - C: 4
  - D: 4
  - E: 6
  - F: 6
  - G: 3
  - H: 3

Sollin’s Algorithm



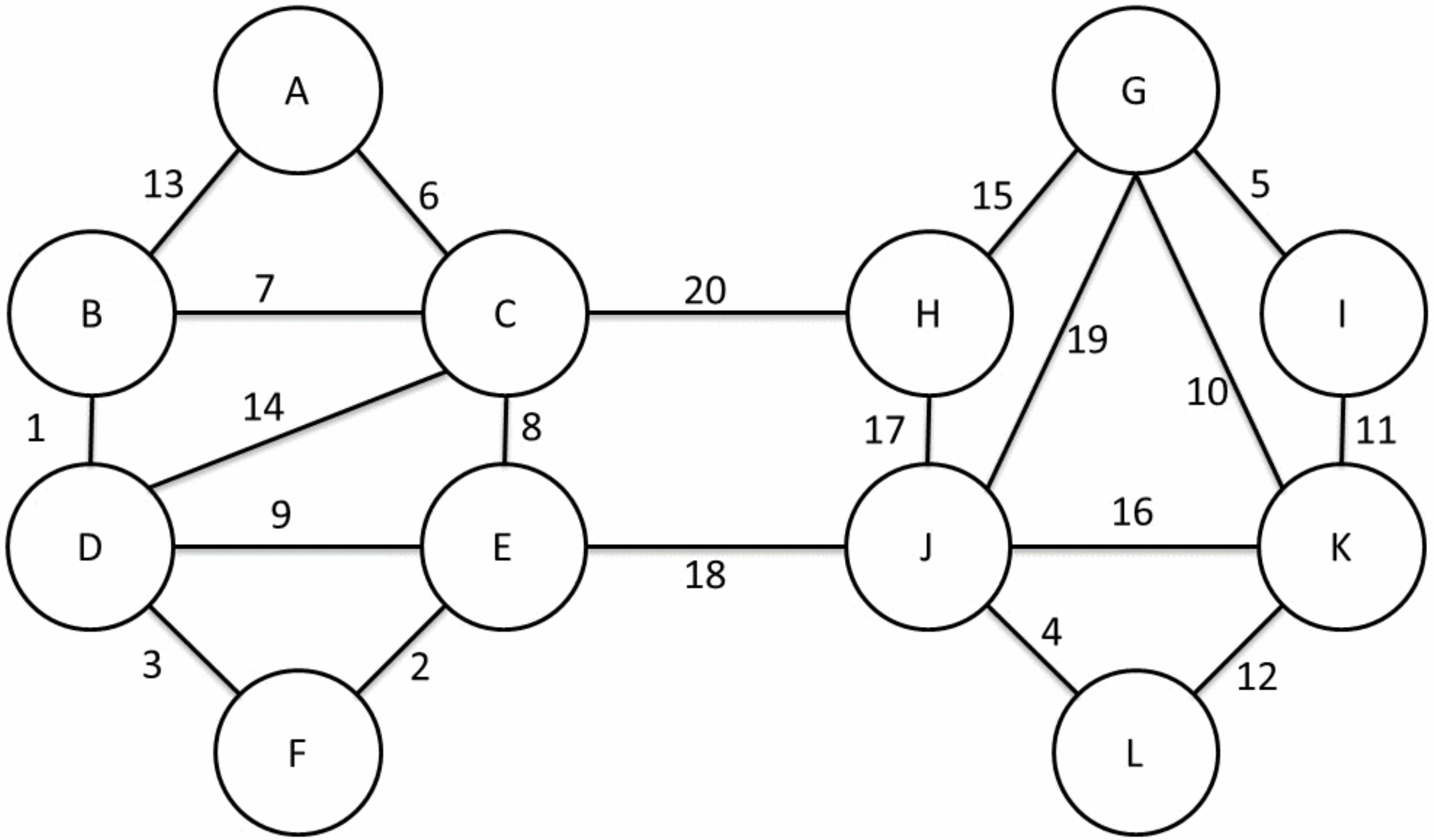
- For each cluster, select the minimum cost inter-cluster edge
  - A: 5
  - B: 5
  - C: 4
  - D: 4
  - E: 6
  - F: 6
  - G: 3
  - H: 3

Sollin's Algorithm



- For each cluster, select the minimum cost inter-cluster edge
  - A, B: 7
  - C, D: 7
  - E, F: 10
  - G, H: 14

Another Example



Time Complexity

- O(E log V)

- Oldest minimum spanning tree algorithm was discovered by Boruuvka in 1926.

References

- Further reading list and references
  - https://www.geeksforgeeks.org/prims-minimum-spanning-tree-mst-greedy-algo-5/
  - https://www.geeksforgeeks.org/kruskals-minimum-spanning-tree-algorithm-greedy-algo-2/?ref=lbp
  - https://algs4.cs.princeton.edu/home/

- Slide credit
  - Jaesik Park
  - Seung-Hwan Baek
  - Jong-Hyeok Lee
  - Carl Kingsford
  - Robert Sedgewick