Minimum Spanning Trees

Instructor: Meng-Fen Chiang

COMPCSI220: WEEK 11

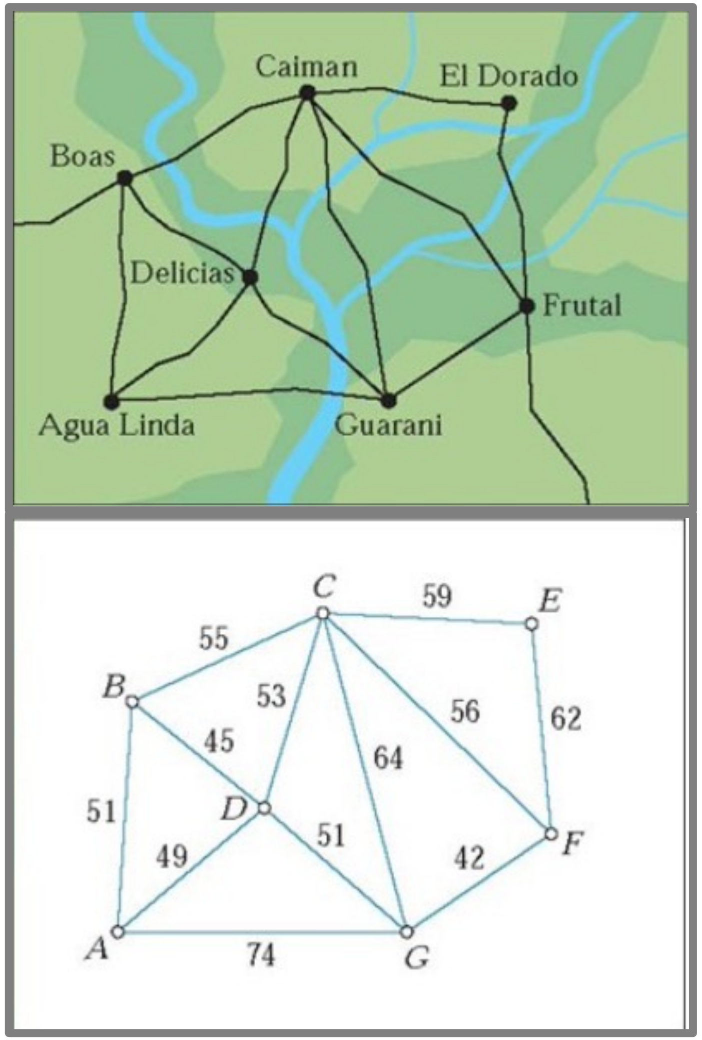
THE UNIVERSITY OF AUCKLAND
Te Whare Wānanga o Tāmaki Makaurau
NEW ZEALAND

Slides adapted from Mark Wilson, Georgy Gimel'farb, Simone Linz, Tanya Gvozdeva, and Kaiqi Zhao

OUTLINE

- Terminology
  - Spanning Trees
  - Minimum Spanning Trees

- Finding minimum Spanning Trees Algorithms
  - Prim’s Algorithm
  - Kruskal's Algorithm
  - Time Complexity Analysis

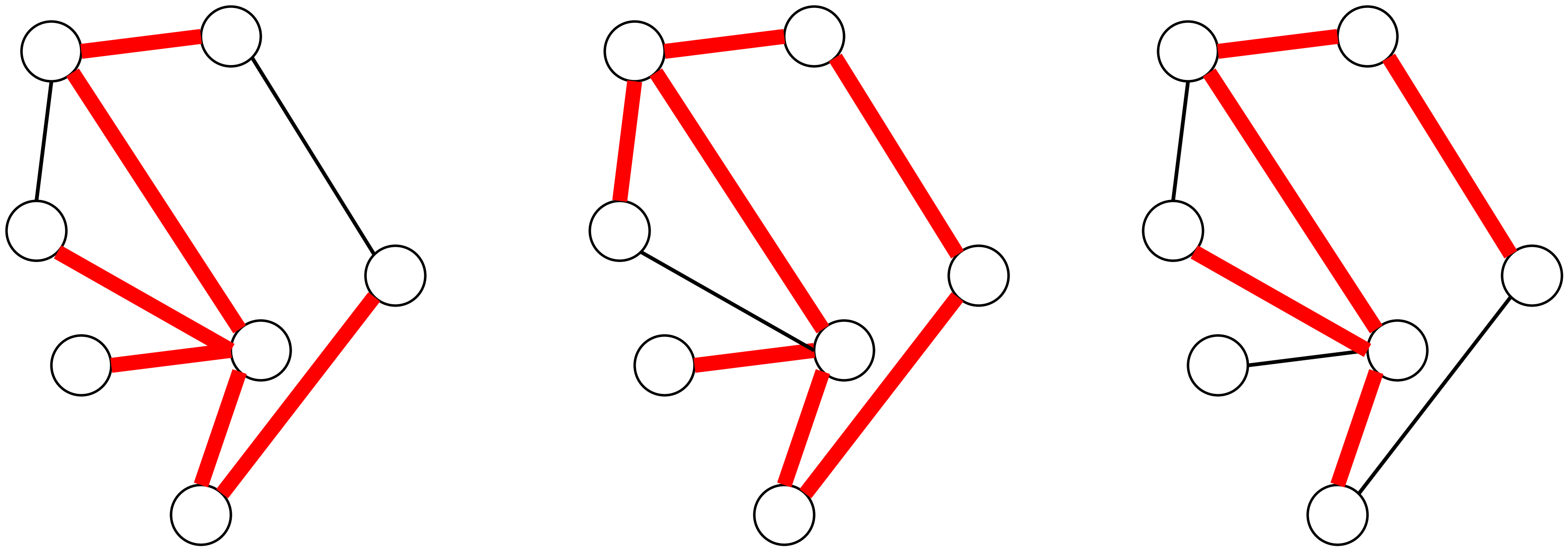


Spanning Trees

- Definition. A spanning tree of a graph G is a connected acyclic graph T such that it contains all the vertices of G, e.g. $\mathrm{V(T)} = \mathrm{V(G)}$, and a subset of edges

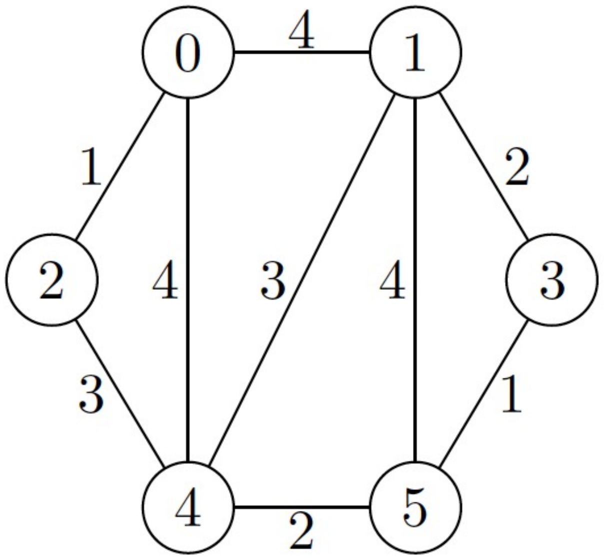
- On a spanning tree T of G, we can walk from any vertex of G to any other vertex of G along exactly one path whose edges are part of the tree T

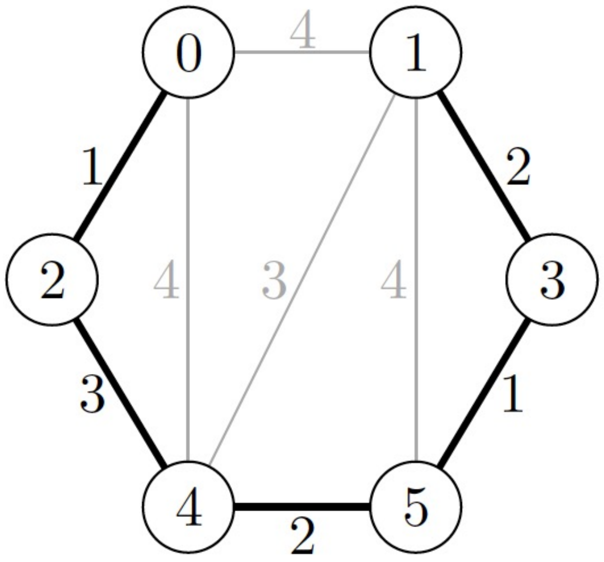
Exercise: Spanning Trees – Yes or No?



Minimum Spanning Tree Problem

- Minimum spanning tree (MST) problem on a weighted connected graph G:
- Find a spanning tree (subgraph containing all vertices that is a tree) of minimum total weight.

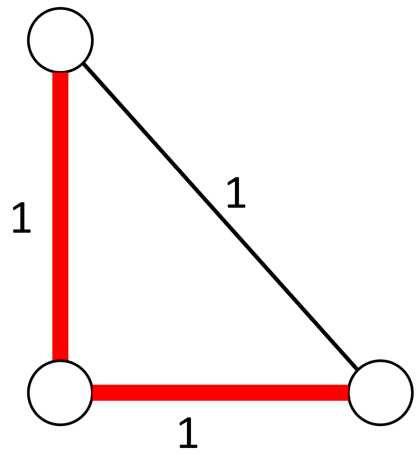


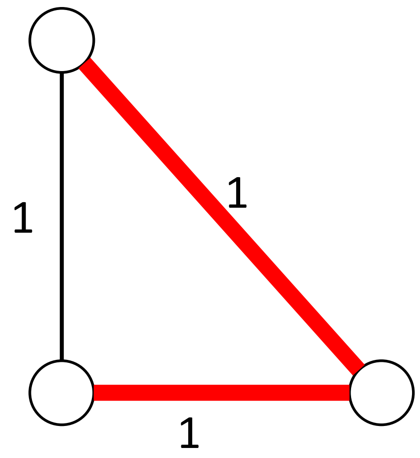


The MST with the total weight 9

Example: Minimum Spanning Trees

- MSTs are not unique





Applications of MSTs

- Design of networks, including computer networks, telecommunications networks, transportation networks, water supply networks, and electrical grids (which they were first invented for, as mentioned above)
- Approximate NP-complete graph optimization (e.g., travel salesman problem)
- Image analysis

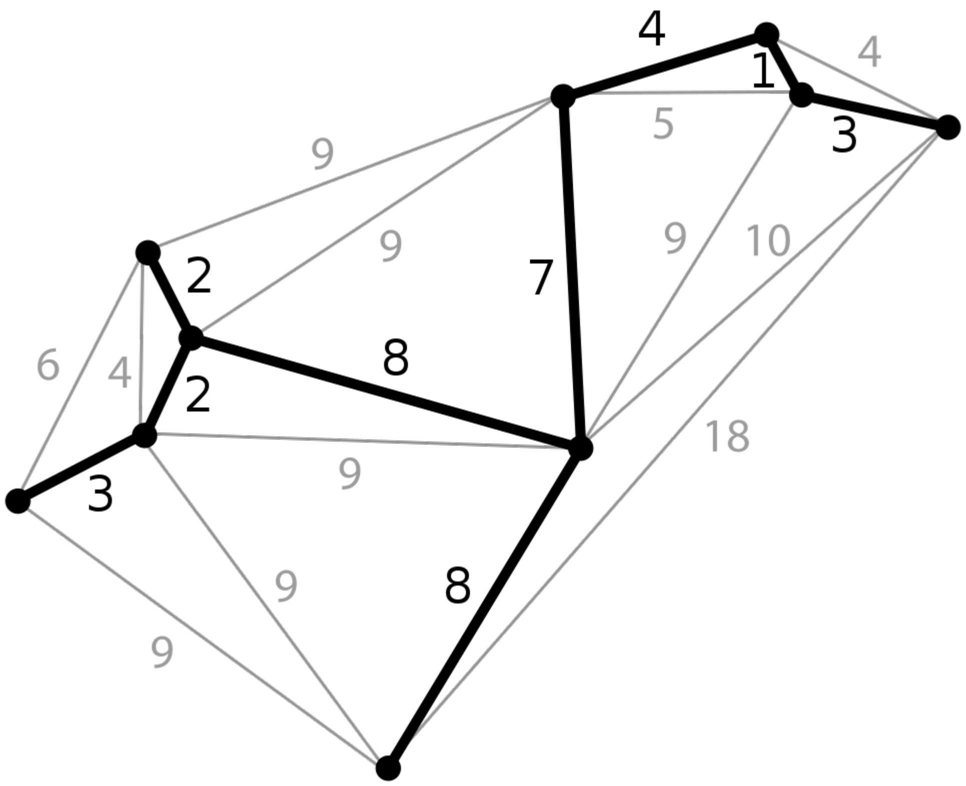


Image from Wikipedia

Minimum Spanning Tree Problem

- Two efficient greedy algorithms presented here: Prim's and Kruskal's.
- Each selects edges in order of increasing weight but avoids creating a cycle.

Prim's

- Prim's algorithm maintains a tree at each stage that grows to span
- Prim's implementation very similar to Dijkstra, runs in time $O((m+n)\log n)$

Kruskal's

- Kruskal maintains a forest whose trees are combined into one spanning tree.
- Kruskal can be implemented to run in time $O(m\log n)$.

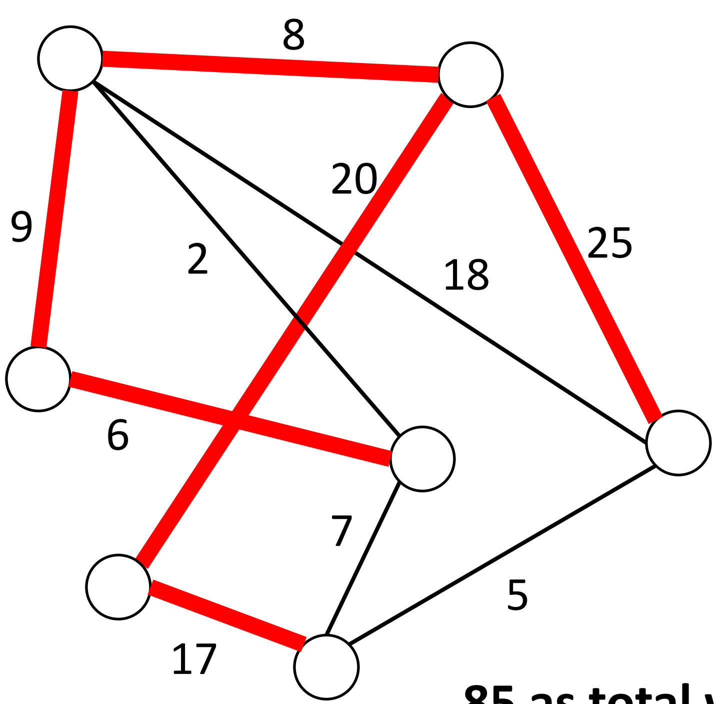
Finding Spanning Trees

- Finding a spanning tree is easy.

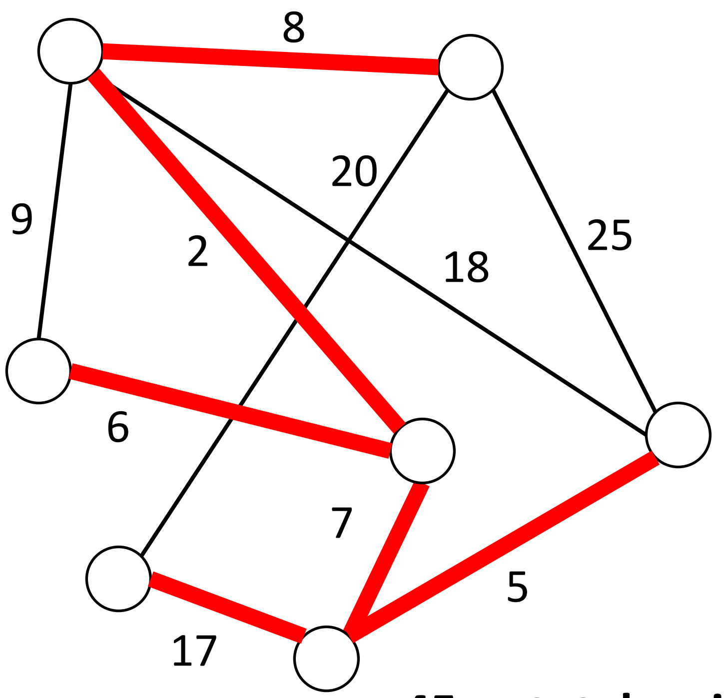
- Steo1: Run DFS or BFS from any of the vertices.
- Step2: If G is connected, there will be only one search tree and this will be a spanning tree.

Spanning Trees in Weighted Graphs (Contd.)

- The total weight of a spanning tree is the sum of the weights of its edges.



85 as total weight

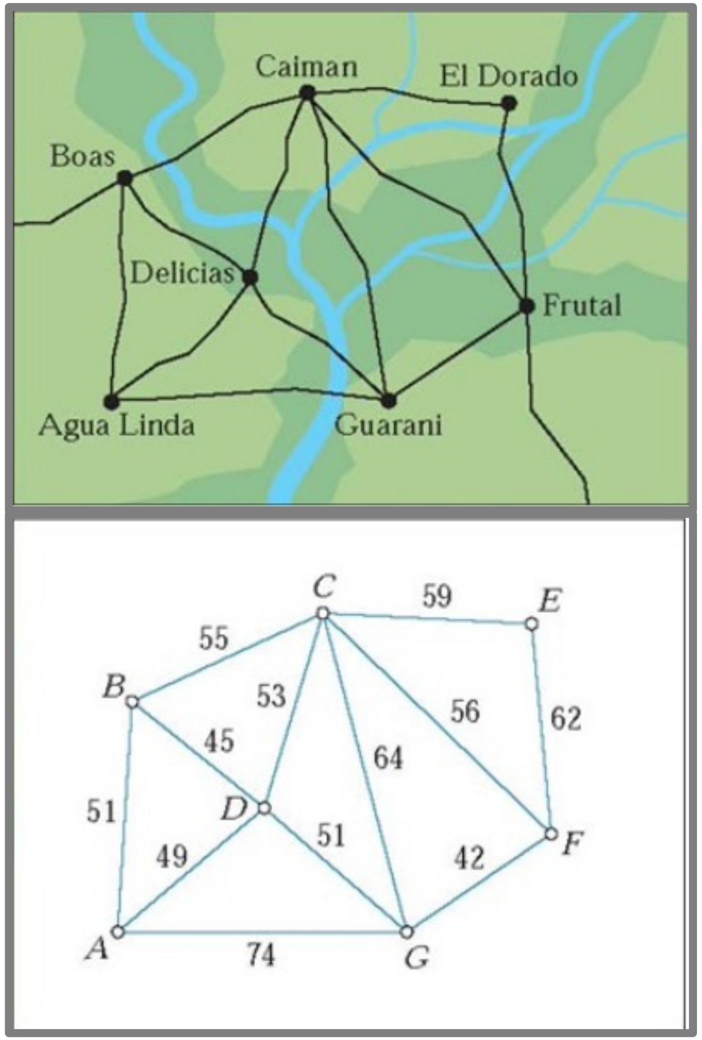


45 as total weight

OUTLINE

- Terminology
  - Spanning Trees
  - Minimum Spanning Trees

- Finding minimum Spanning Trees Algorithms
  - Prim’s Algorithm
  - Kruskal's Algorithm
  - Time Complexity Analysis



Prim’s Algorithm

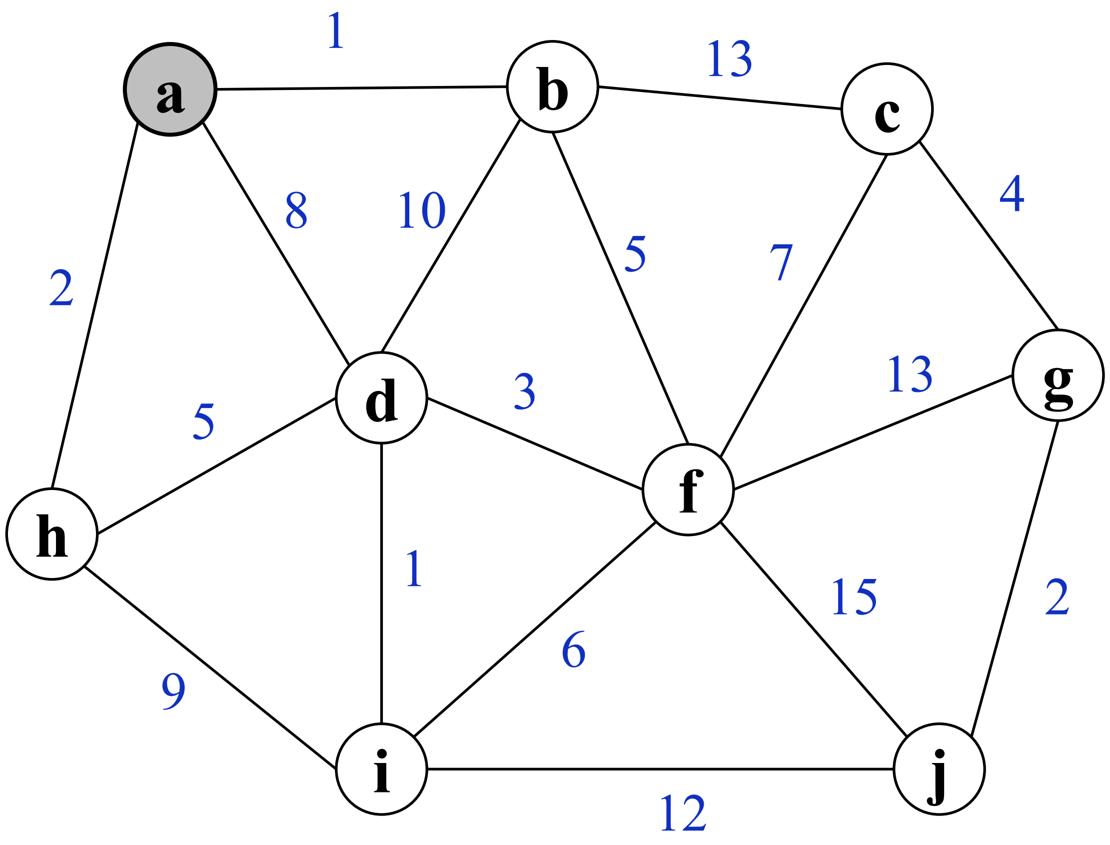
- Prim's algorithm is a modified version of Dijkstra's.
- We simply pick one of the vertices in G as the source.
- We need to keep track of the edges used to reach a vertex.

Algorithm 1 Prim's algorithm.

```
function PRIM(weighted graph(G, c); vertex s ∈ V(G))
    priority queue Q
    array colour[0..n − 1], pred[0..n − 1]
    for u ∈ V(G) do
        colour[u] ← WHITE; pred[u] ← null
    colour[s] ← GREY
    Q.insert(s, 0)
    while not Q.isEmpty() do
        u ← Q.peek()
        for each x adjacent to u do
            t ← c(u, x)
            if colour[x] = WHITE then
                colour[x] ← GREY; pred[x] ← u
                Q.insert(x, t)
            else if colour[x] = GREY and Q.getKey(x) > t then
                Q.decreaseKey(x, t); pred[x] ← u
        Q.delete()
        colour[u] ← BLACK
    return pred
```

Illustrating Prim's algorithm

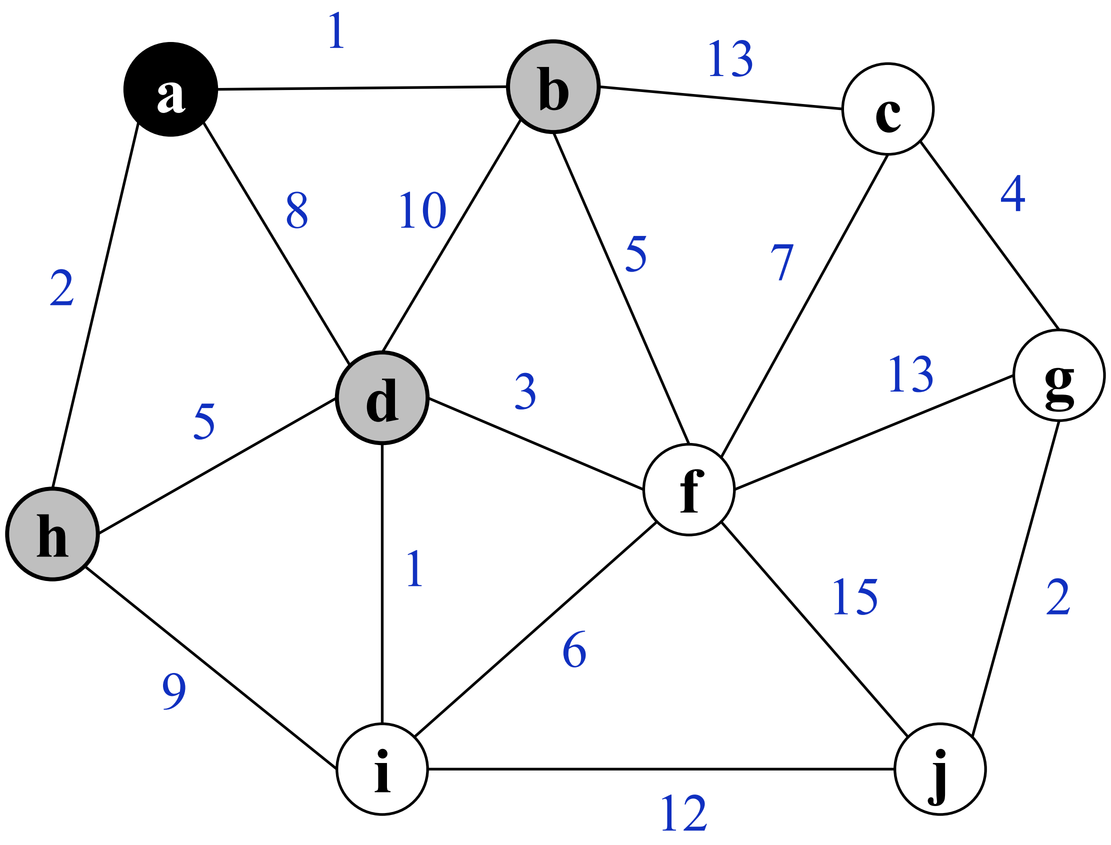
Priority Queue Q:
(a, 0)



Pred:
a: null
b: null
c: null
d: null
f: null
g: null
h: null
i: null
j: null

Illustrating Prim's algorithm

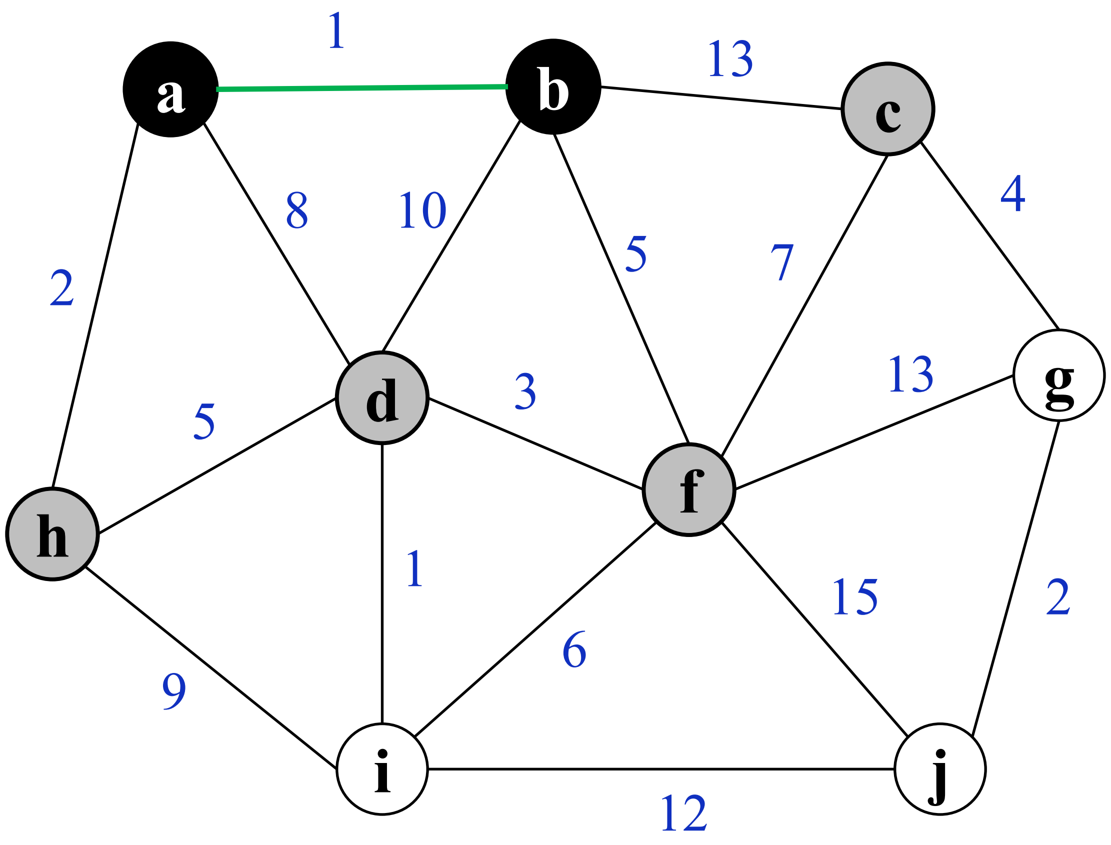
Priority Queue Q:
(b, 1)
(d, 8)
(h, 2)



Pred:
a: null
b: a
c: null
d: a
f: null
g: null
h: a
i: null
j: null

Illustrating Prim's algorithm

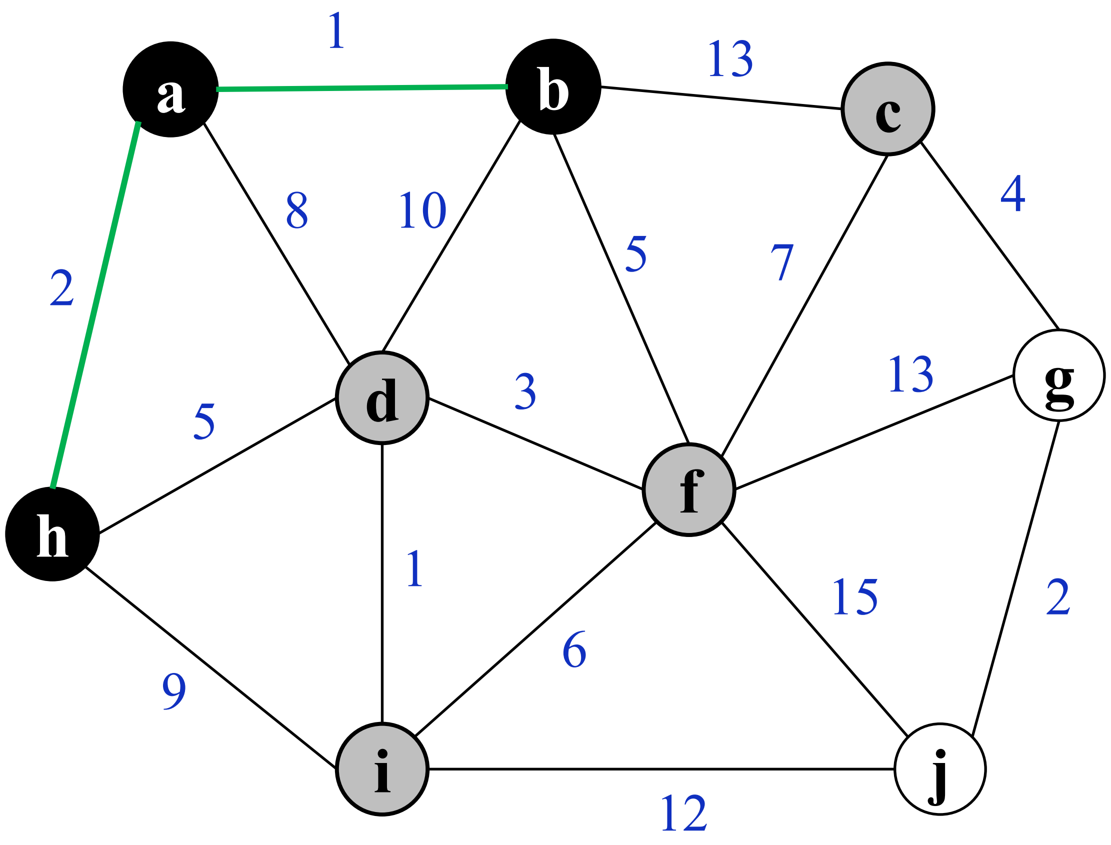
Priority Queue Q:
(d, 8)
(h, 2)
(c, 13)
(f, 5)



Pred:
a: null
b: a
c: b
d: a
f: b
g: null
h: a
i: null
j: null

Illustrating Prim's algorithm

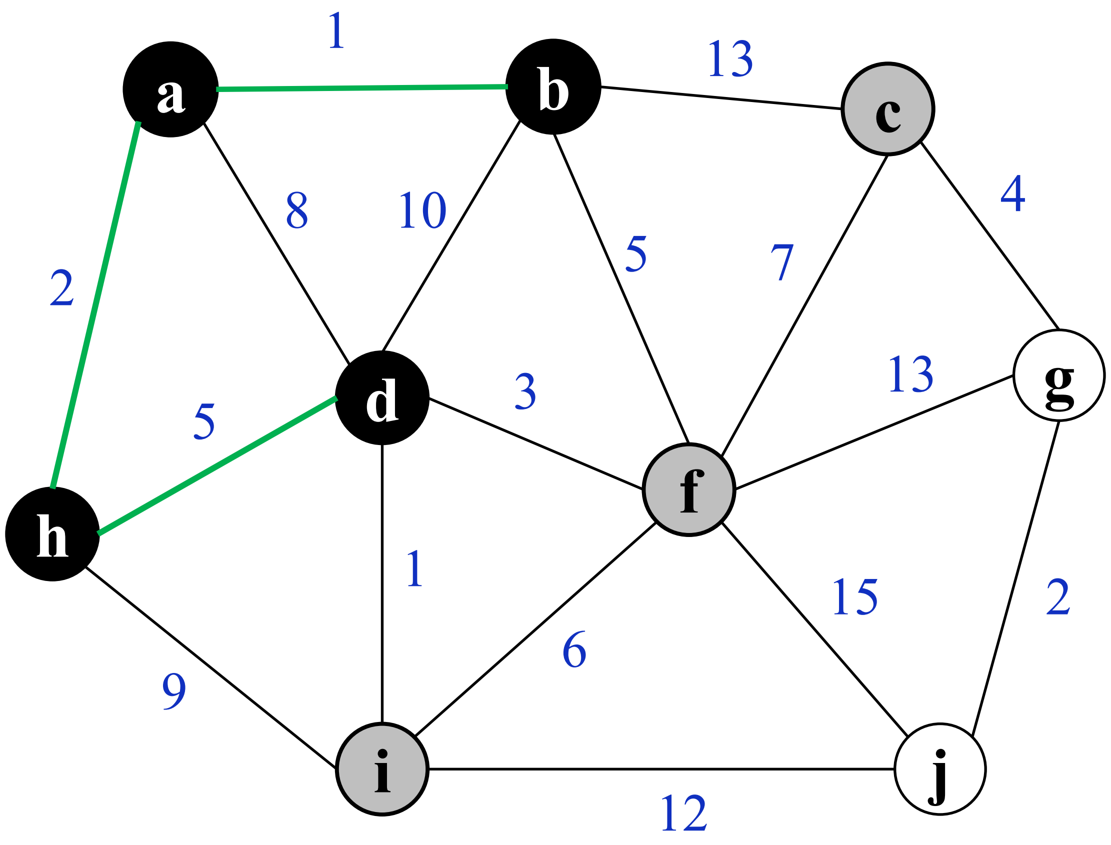
Priority Queue Q:
(d, 5)
(c, 13)
(f, 5)
(i, 9)



Pred:
a: null
b: a
c: b
d: h
f: b
g: null
h: a
i: h
j: null

Illustrating Prim's algorithm

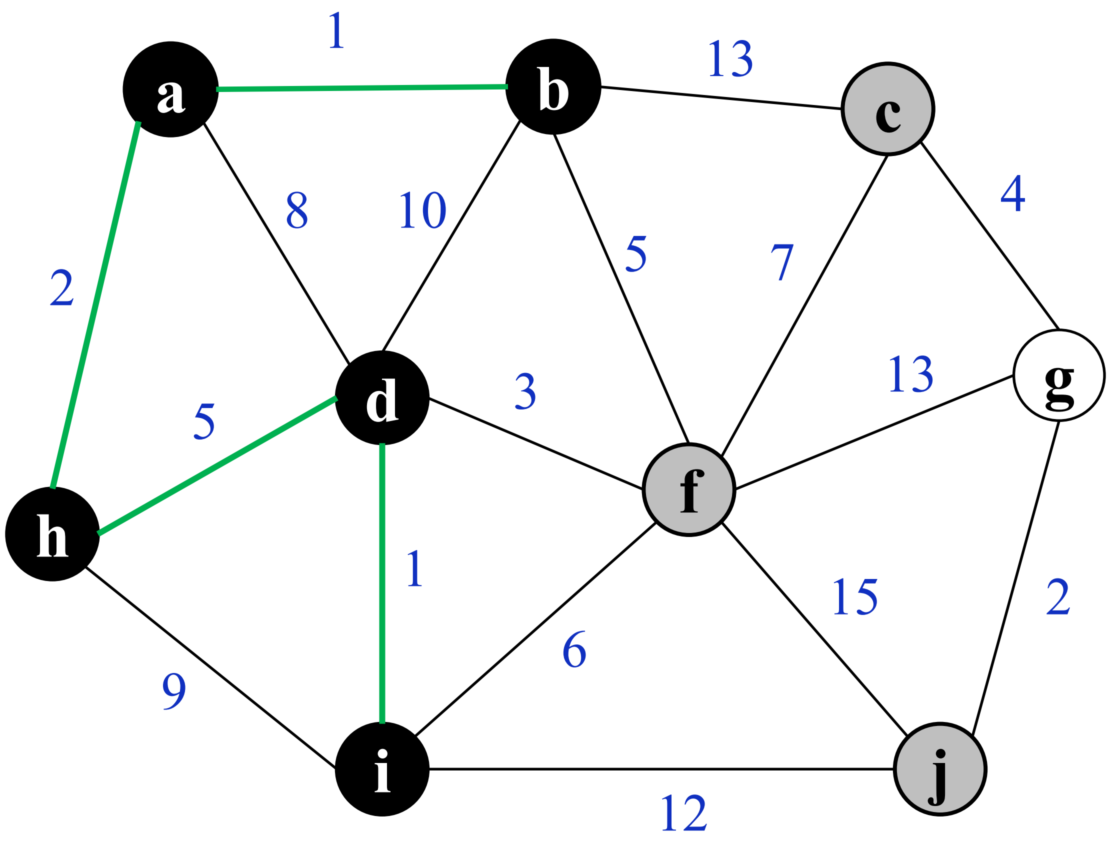
Priority Queue Q:
(c, 13)
(f, 3)
(i, 1)



Pred:
a: null
b: a
c: b
d: h
f: d
g: null
h: a
i: d
j: null

Illustrating Prim's algorithm

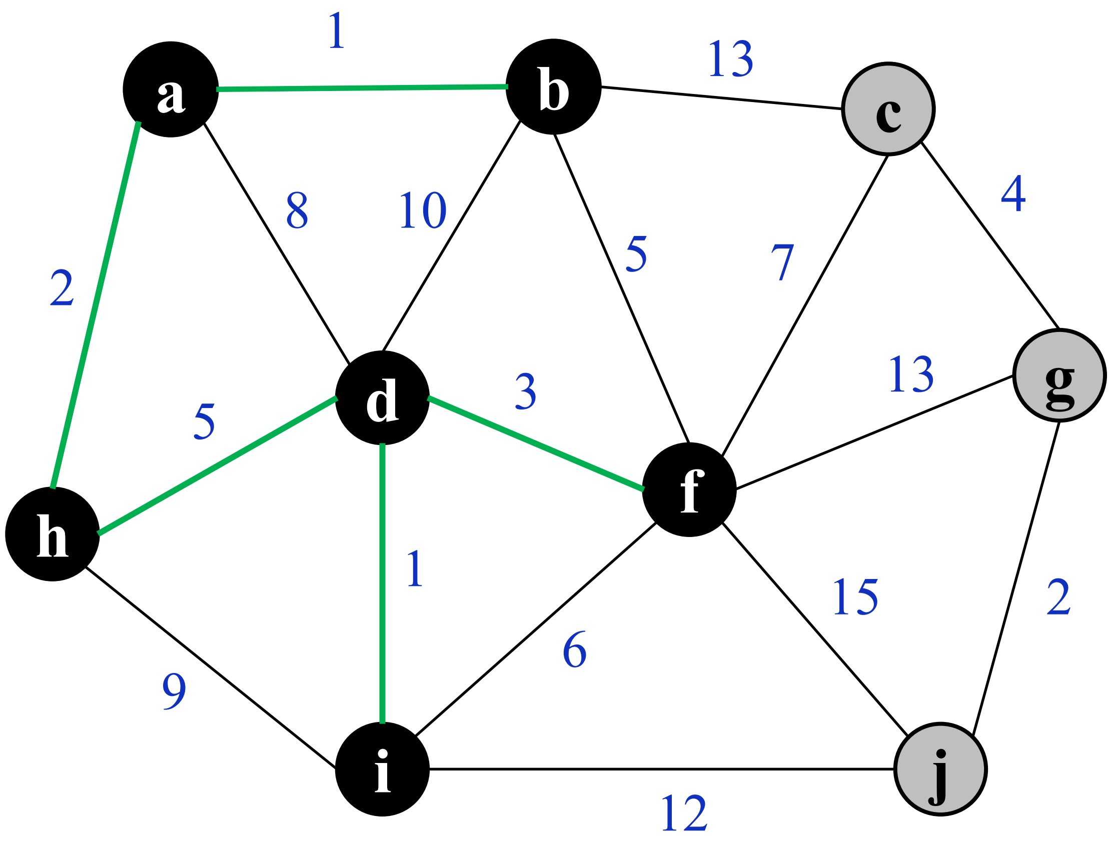
Priority Queue Q:
(c, 13)
(f, 3)
(j, 12)



Pred:
a: null
b: a
c: b
d: h
f: d
g: null
h: a
i: d
j: i

Illustrating Prim's algorithm

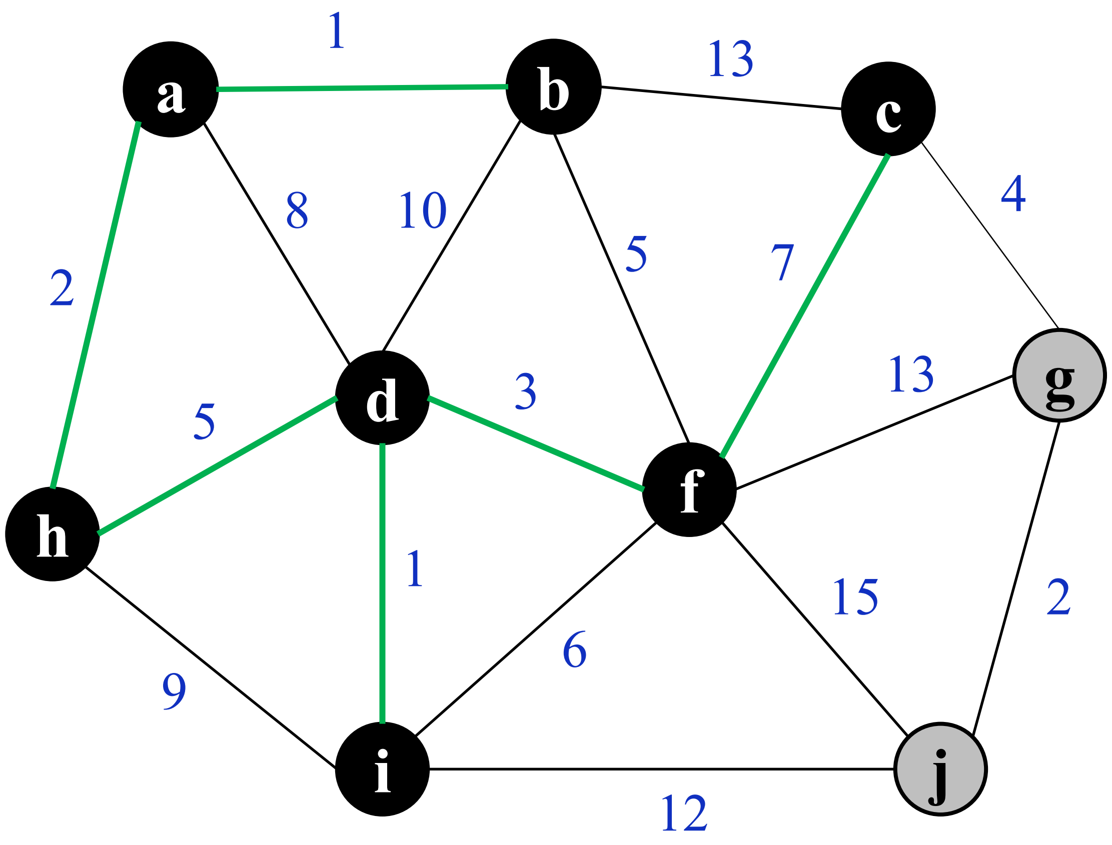
Priority Queue Q:
(c, 7)
(j, 12)
(g, 13)



Pred:
a: null
b: a
c: f
d: h
f: d
g: f
h: a
i: d
j: i

Illustrating Prim's algorithm

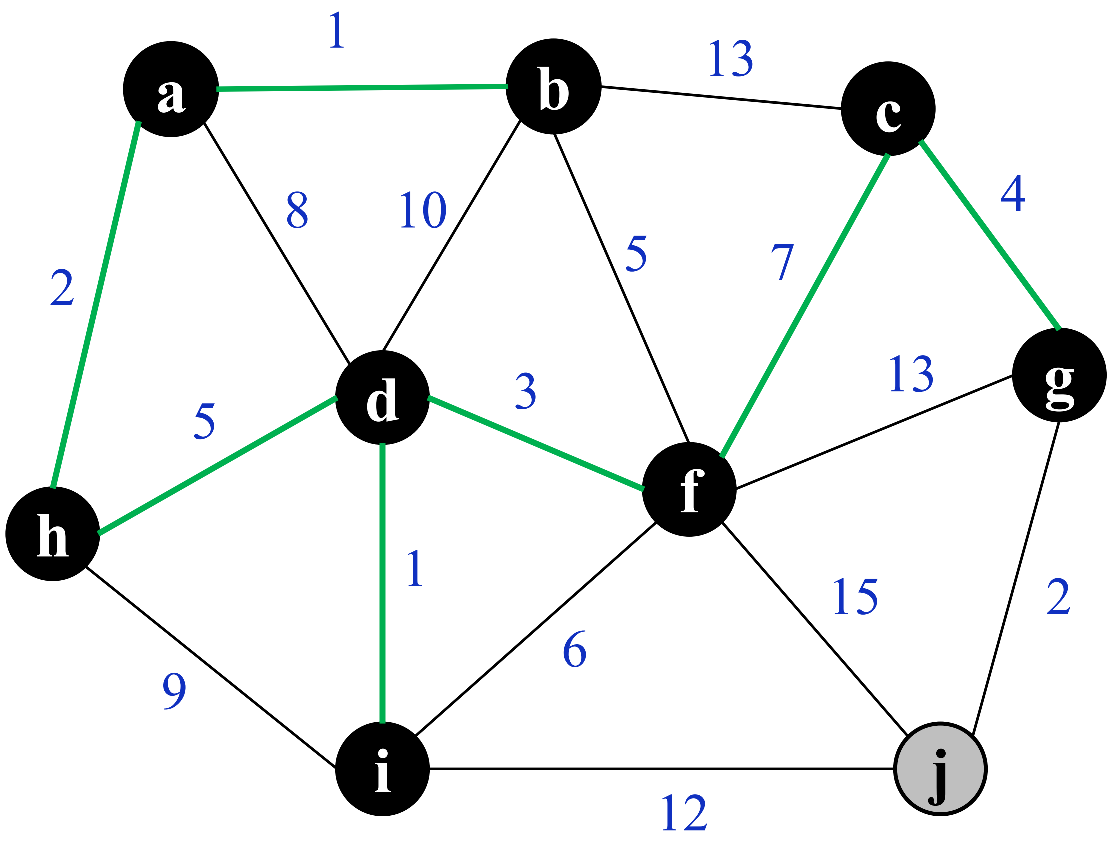
Priority Queue Q:
(j, 12)
(g, 4)



Pred:
a: null
b: a
c: f
d: h
f: d
g: c
h: a
i: d
j: i

Illustrating Prim's algorithm

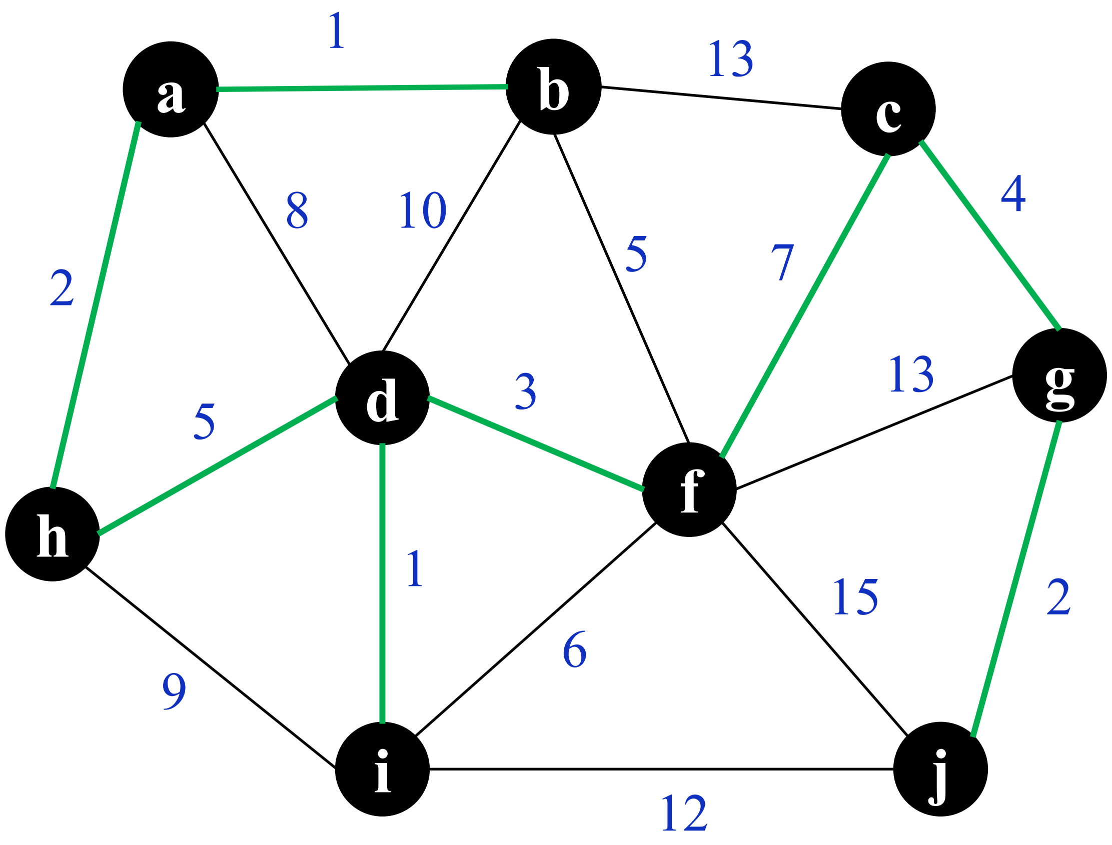
Priority Queue Q:
(j, 2)



Pred:
a: null
b: a
c: f
d: h
f: d
g: c
h: a
i: d
j: g

Illustrating Prim's algorithm

Priority Queue Q:



Pred:

a: null

b: a

c: f

d: h

f: d

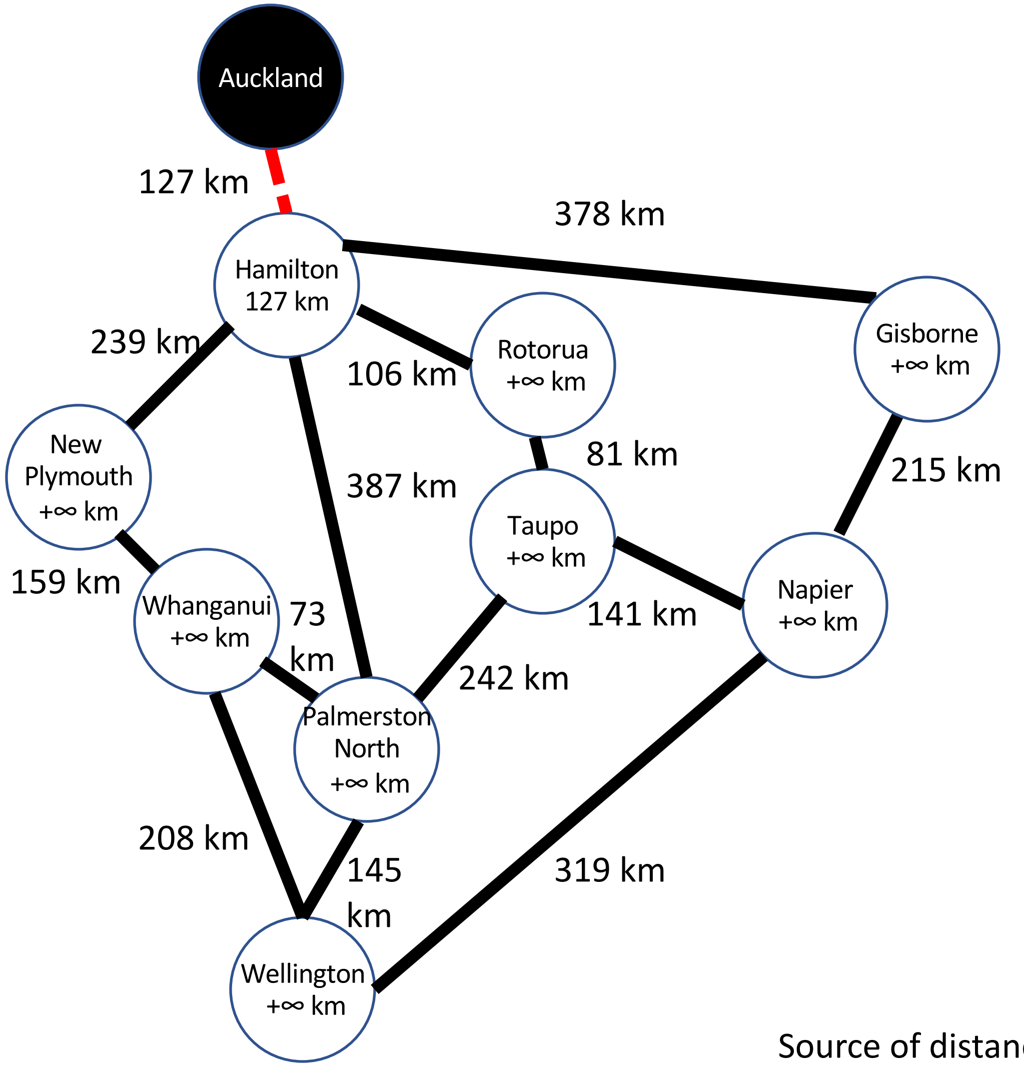
g: c

h: a

i: d

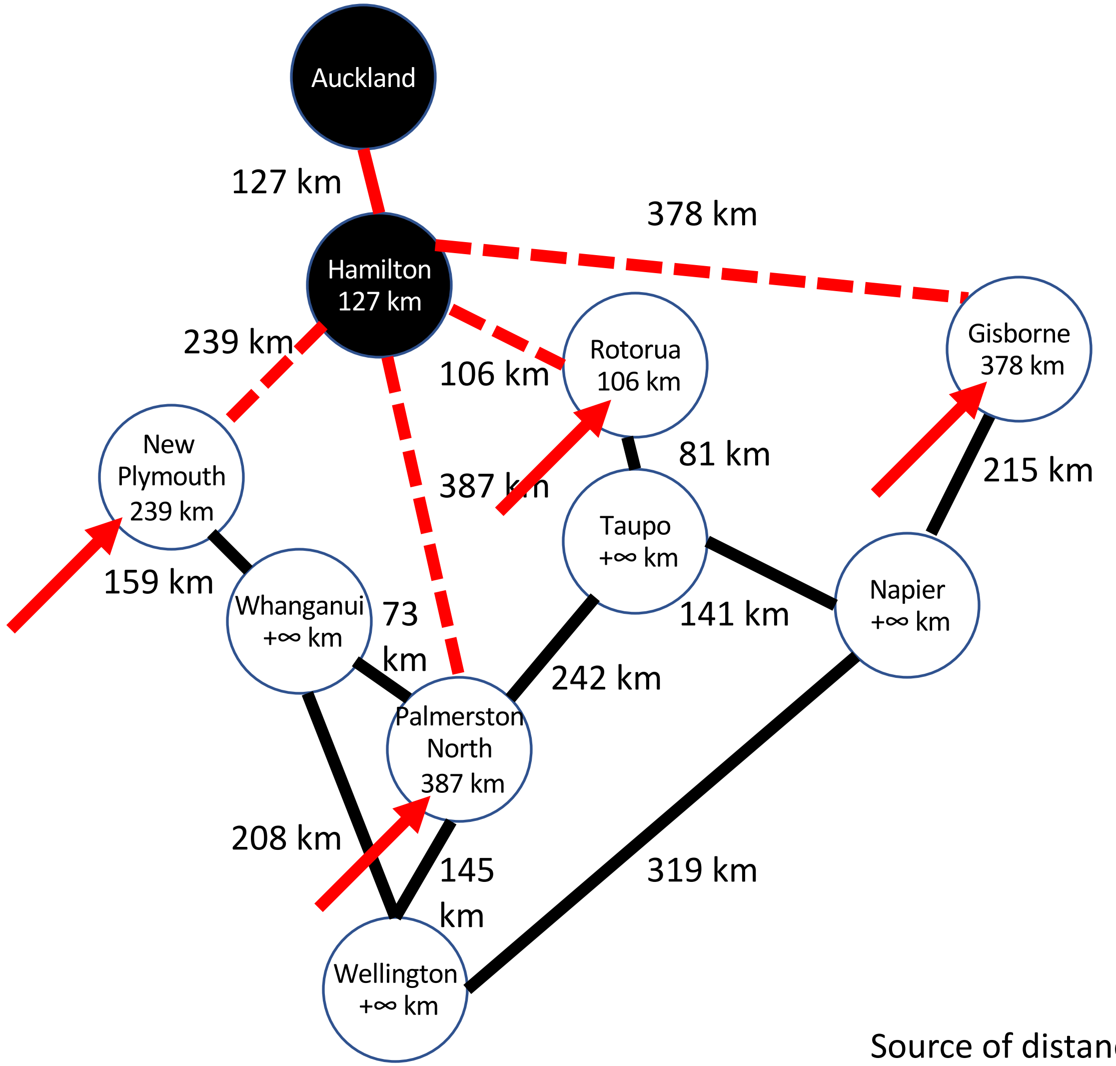
j: g

Example: Prim’s Algorithm at Work



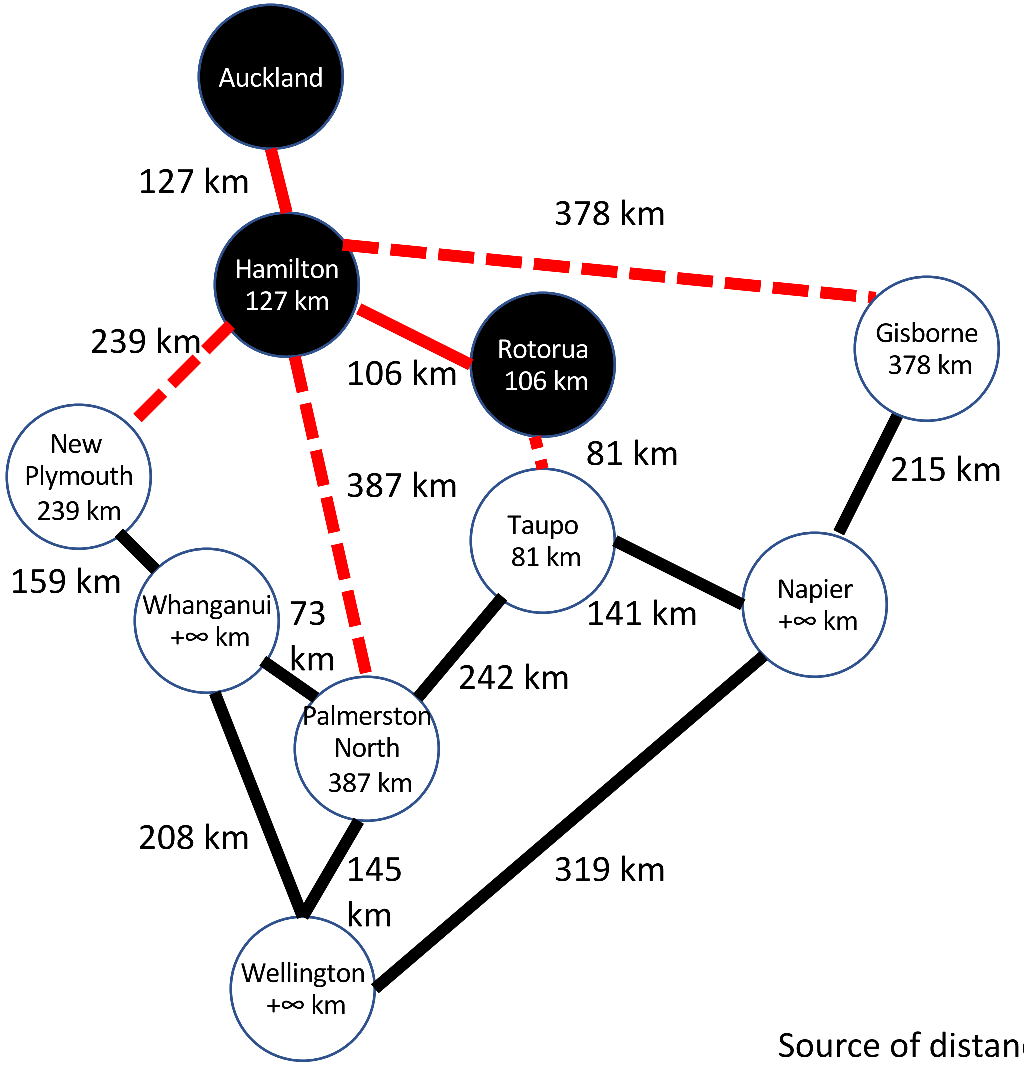
Source of distances: Google Maps

Example: Prim's Algorithm at Work



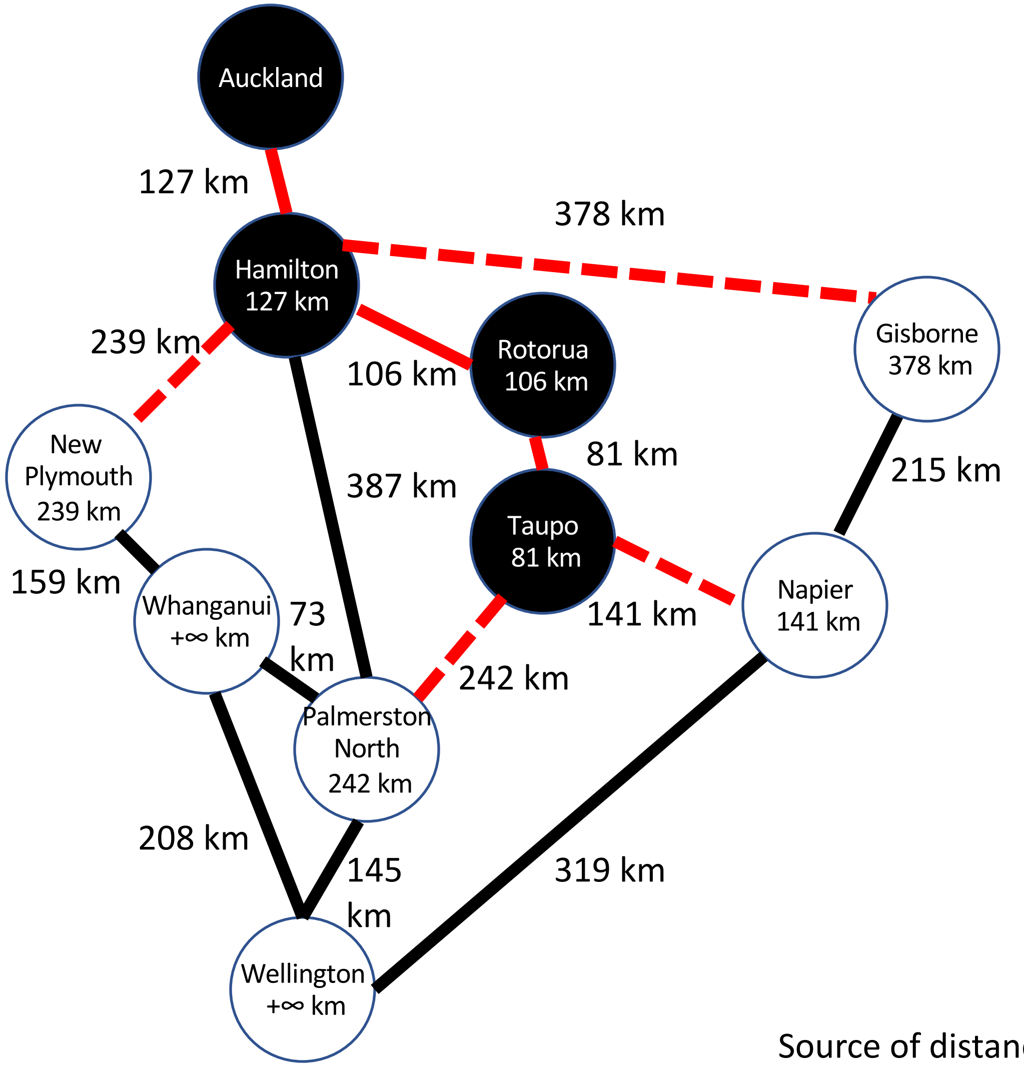
Source of distances: Google Maps

Example: Prim’s Algorithm at Work



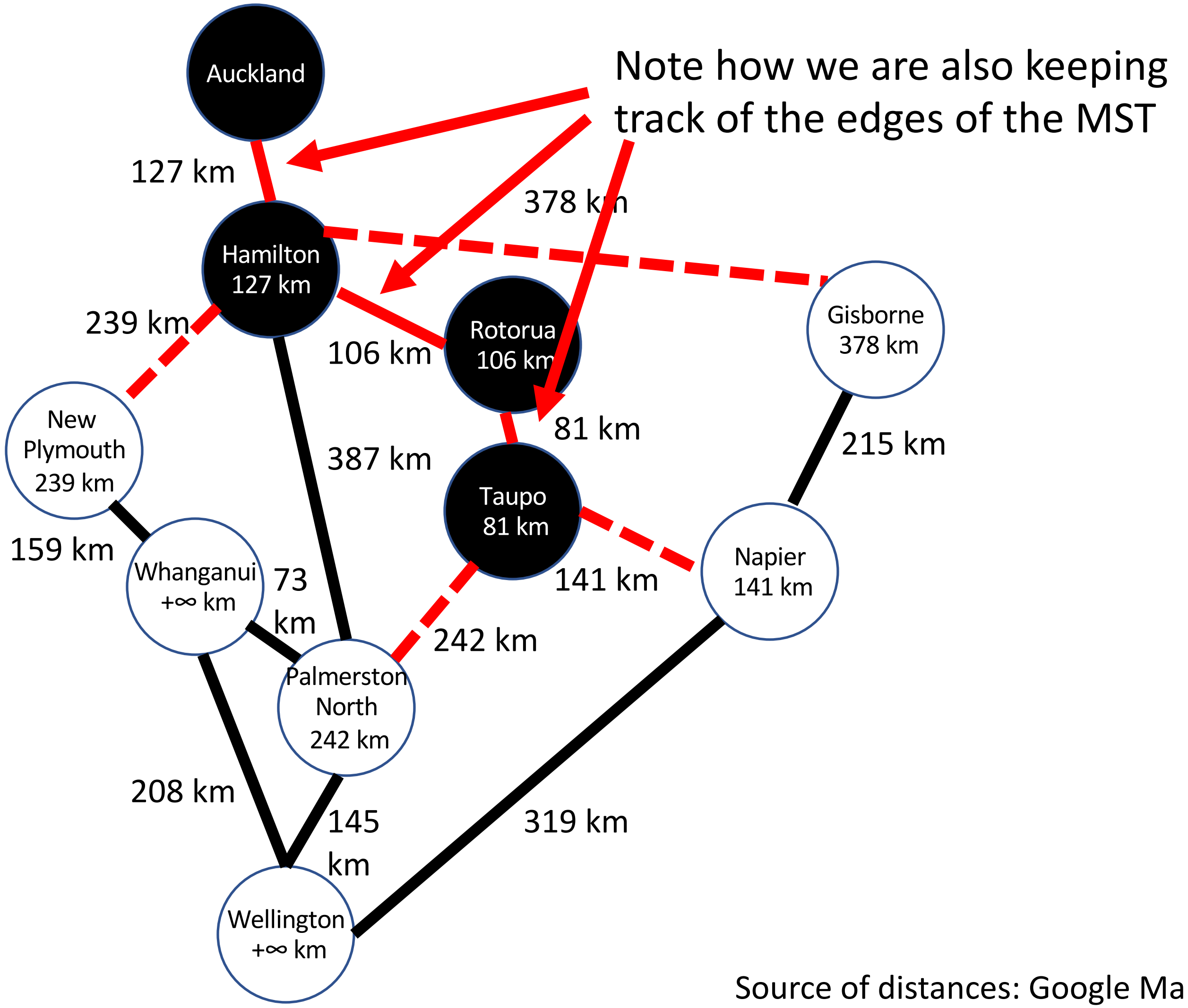
Source of distances: Google Maps

Example: Prim's Algorithm at Work



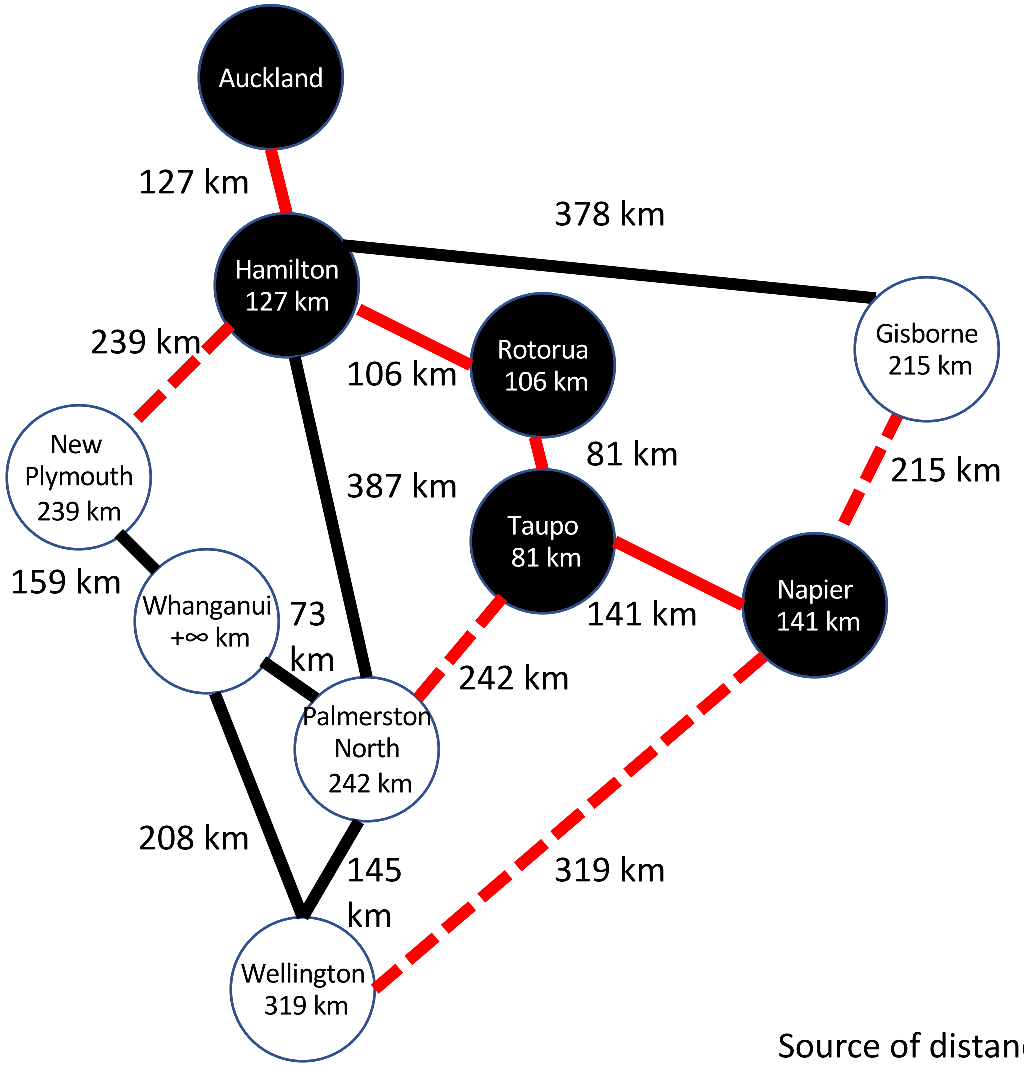
Source of distances: Google Maps

Example: Prim's Algorithm at Work



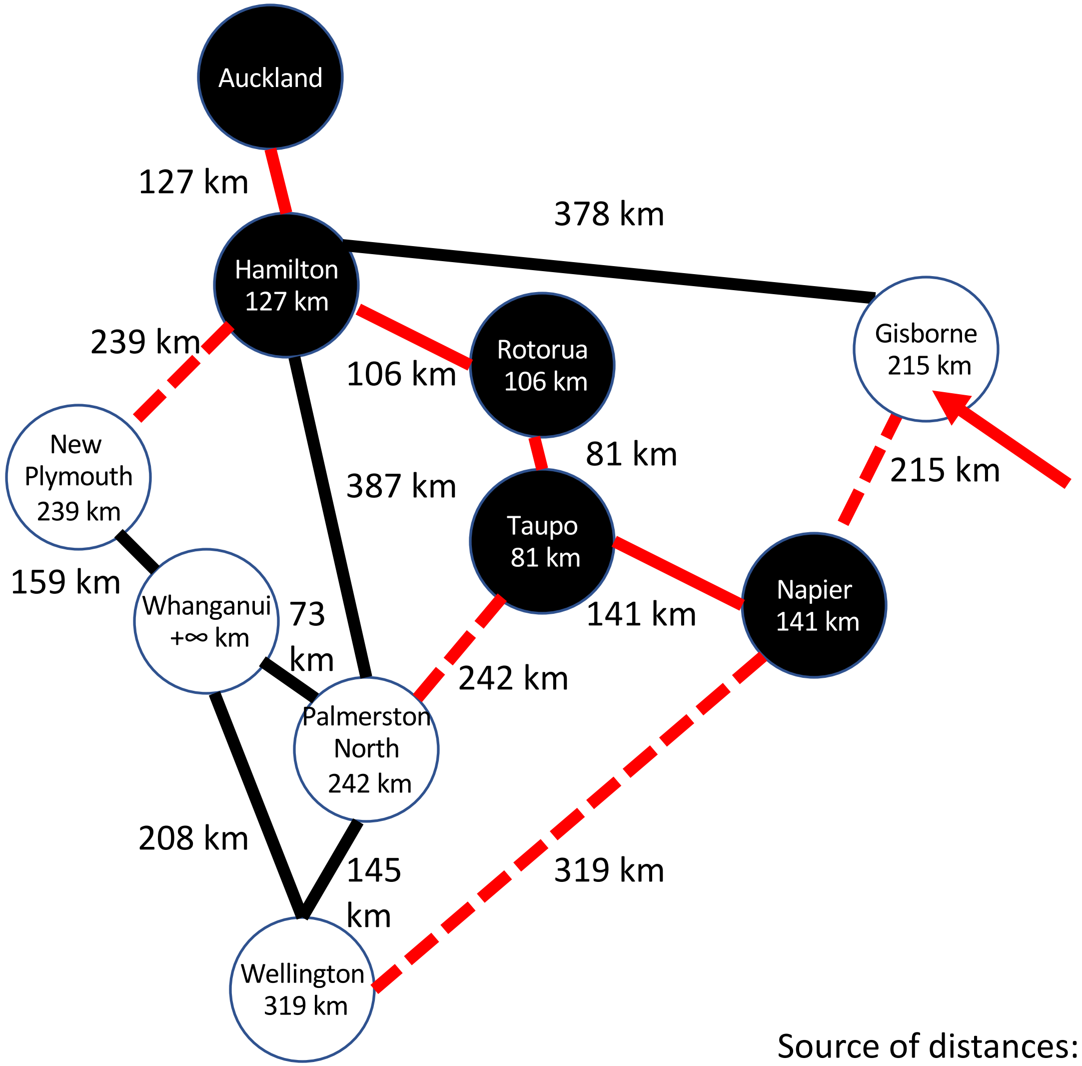
Source of distances: Google Maps

Example: Prim's Algorithm at Work



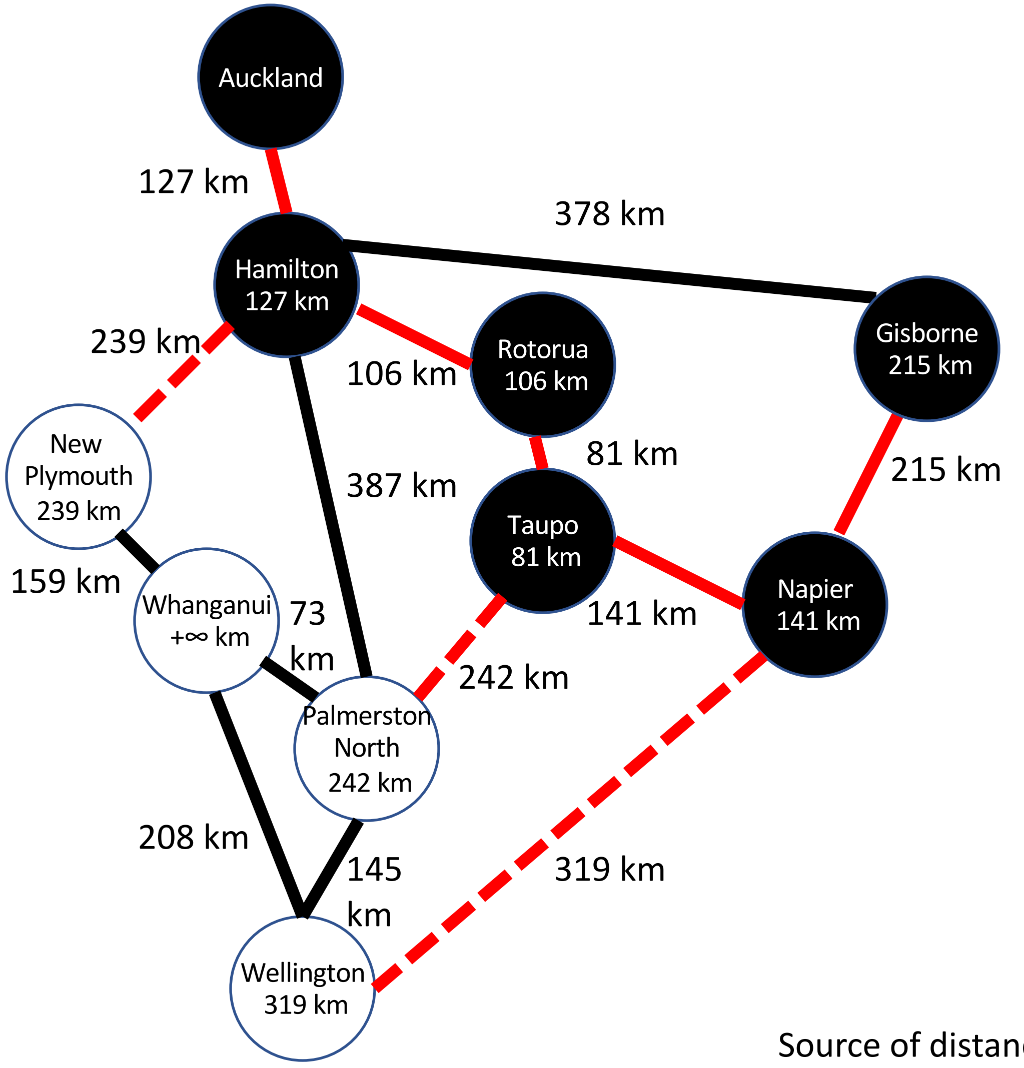
Source of distances: Google Maps

Example: Prim's Algorithm at Work



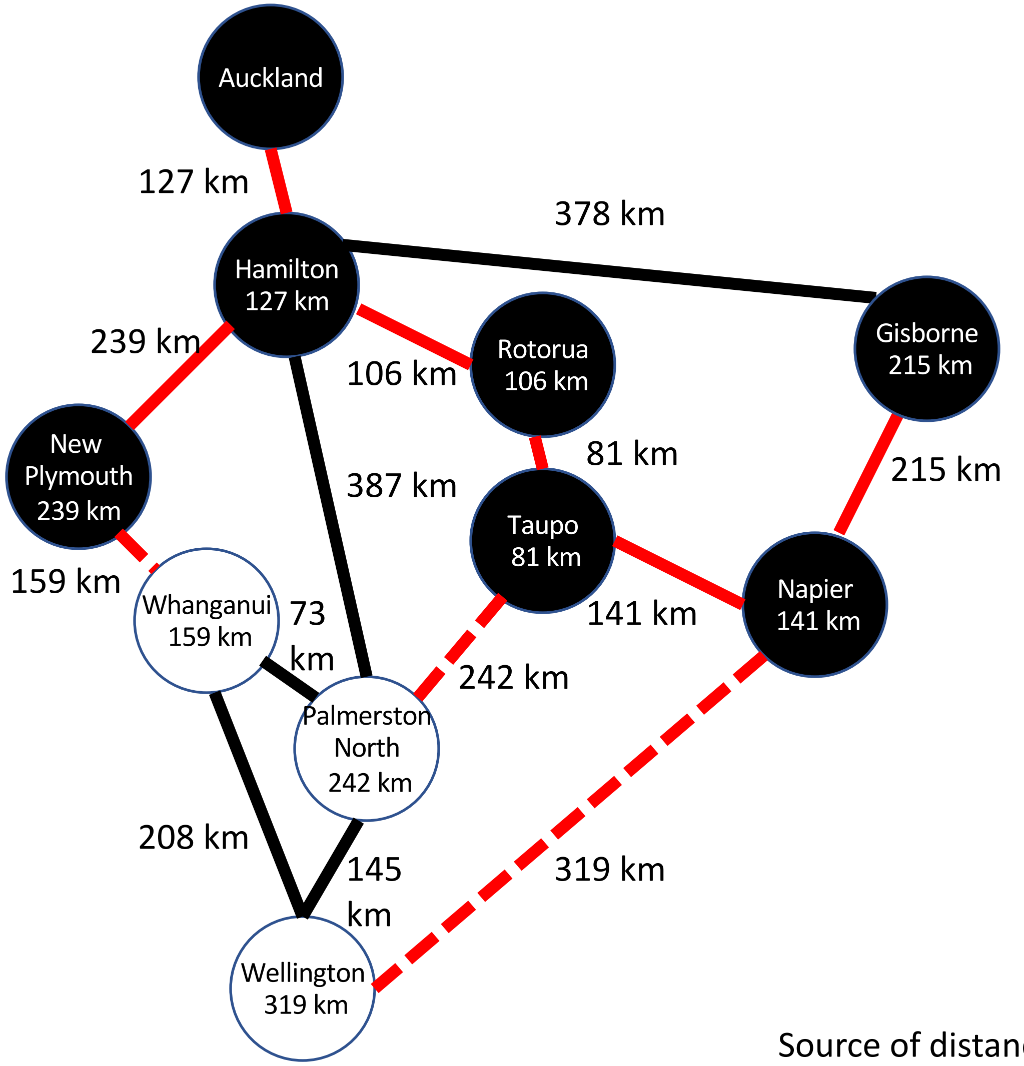
Source of distances: Google Maps

Example: Prim's Algorithm at Work



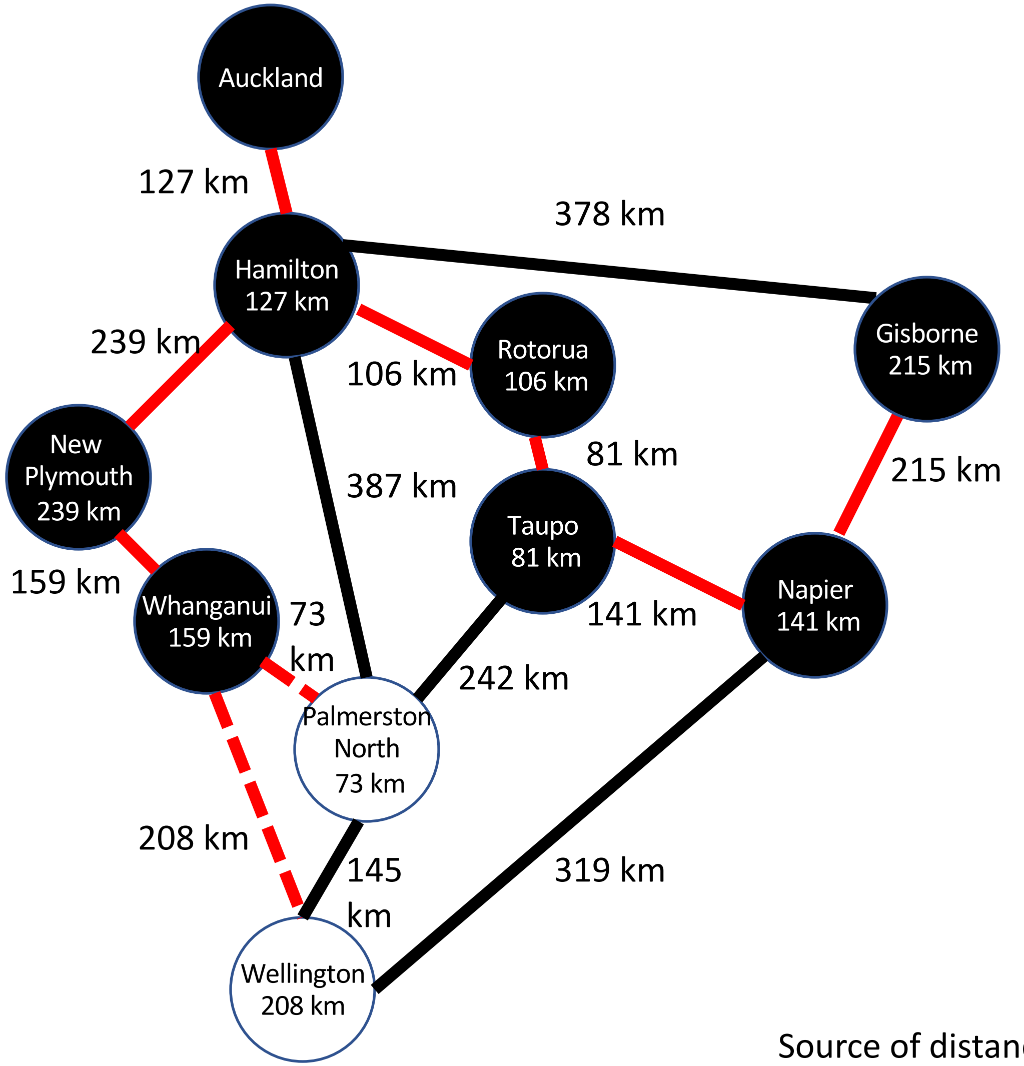
Source of distances: Google Maps

Example: Prim’s Algorithm at Work



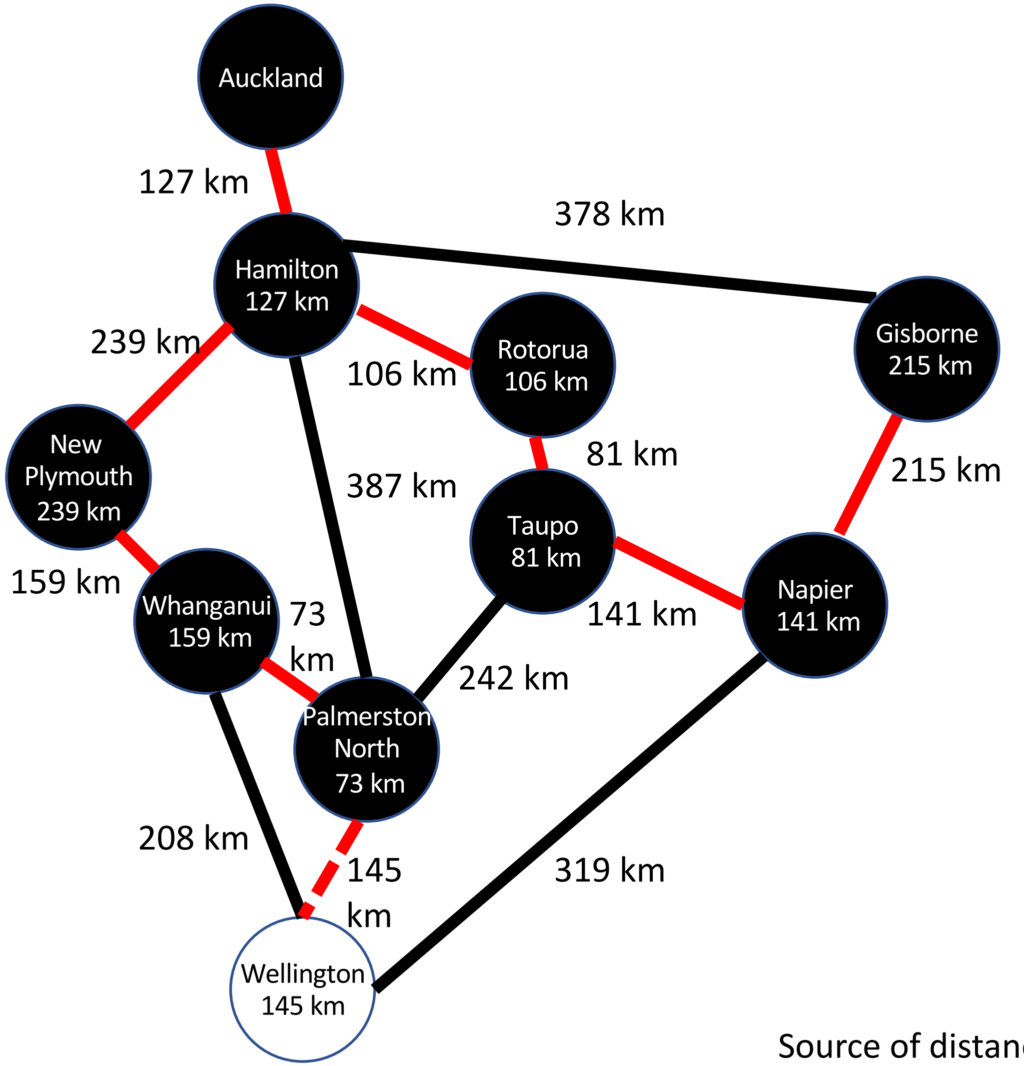
Source of distances: Google Maps

Example: Prim's Algorithm at Work



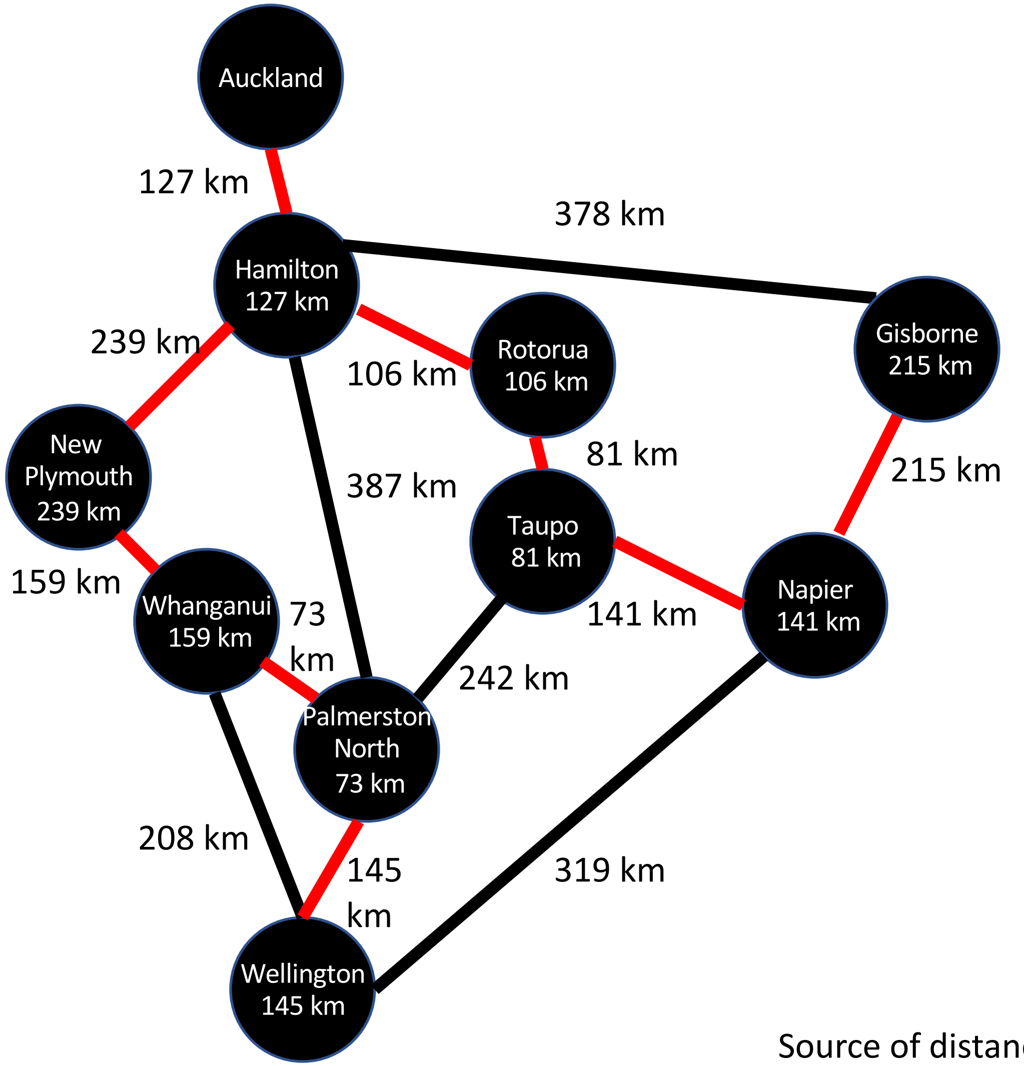
Source of distances: Google Maps

Example: Prim’s Algorithm at Work



Source of distances: Google Maps

Example: Prim's Algorithm at Work



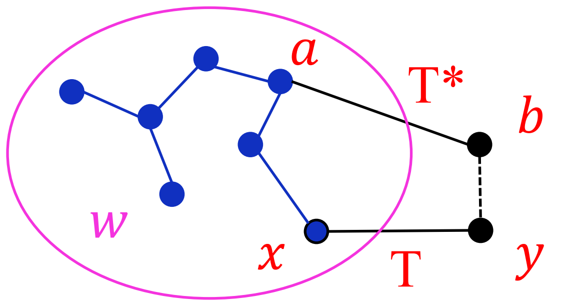
Source of distances: Google Maps

Time Complexity Analysis

- Prim's algorithm is essentially the same as Dijkstra's
- The time complexity therefore is the same as Dijkstra's.

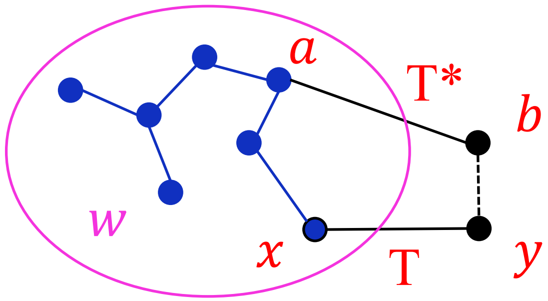
Correctness of Prim's Algorithm

- Proof by **contradiction**.
- Let T be a spanning tree that is obtained by Prim's algorithm and assume that T is not of minimum weight. Let $e_1, e_2, \dots, e_{n-1}$ be the edges of T chosen in order. Let T* be an MST with edges $f_1, f_2, \dots, f_{n-1}$.
- We may assume that T* is chosen such that $i \in \{1,2,\dots,n-1\}$ with $e_j = f_j$ for all $j < i$ is maximum over all choices for T*. Let w be the set of vertices selected in T before e_i is selected, and (x, y) be the first edge in T but not in T*. Let $E = E(T^*) \cup \{(x,y) - (a,b)\}$ be the edge set of a spanning tree.



Correctness of Prim's Algorithm (Contd.)

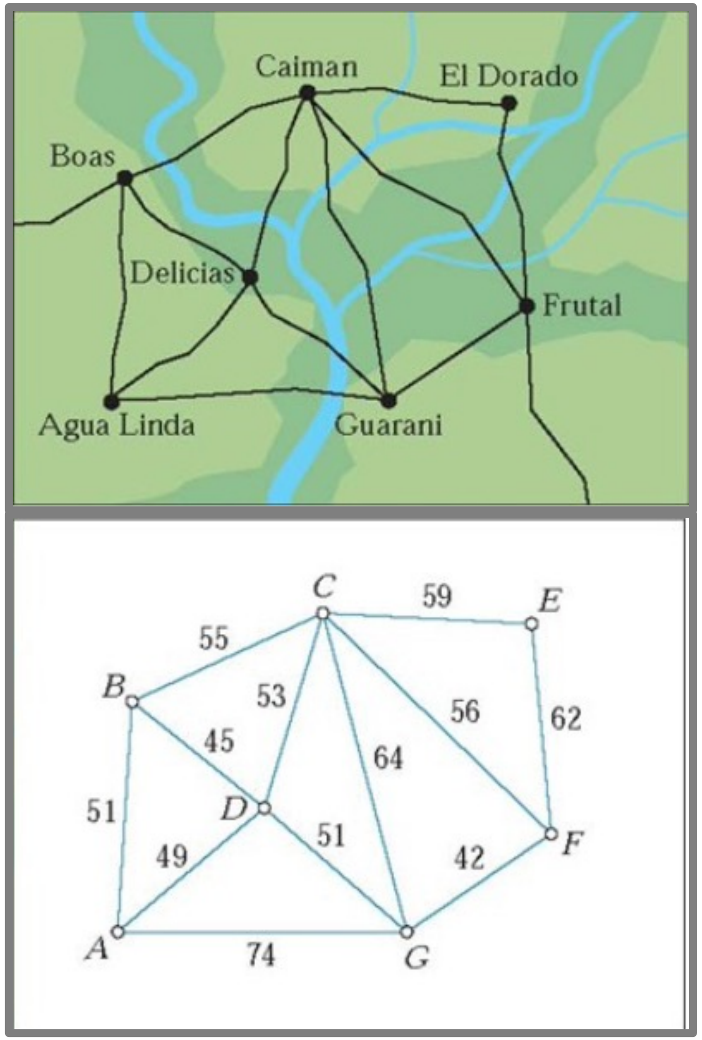
- Case 1: $c(a,b)>c(x,y)$. Then E has a smaller weight than T*.
- Case 2: $c(a,b)<c(x,y)$. Then the algorithm chooses (a,b) and not (x,y).
- Case 3: $c(a,b)=c(x,y)$. Then there exists an MST whose first i edges are identical to T.



OUTLINE

- Terminology
  - Spanning Trees
  - Minimum Spanning Trees

- Finding minimum Spanning Trees Algorithms
  - Prim’s Algorithm
  - Kruskal's Algorithm
  - Time Complexity Analysis



Kruskal’s Algorithm

- Prim’s builds an MST using black vertices
- Kruskal’s builds a forest that then converges to a MST
- Both algorithms build the tree avoiding to add cycles

Kruskal’s Algorithm (Contd.)

- Step1: First sort the edges by their weights

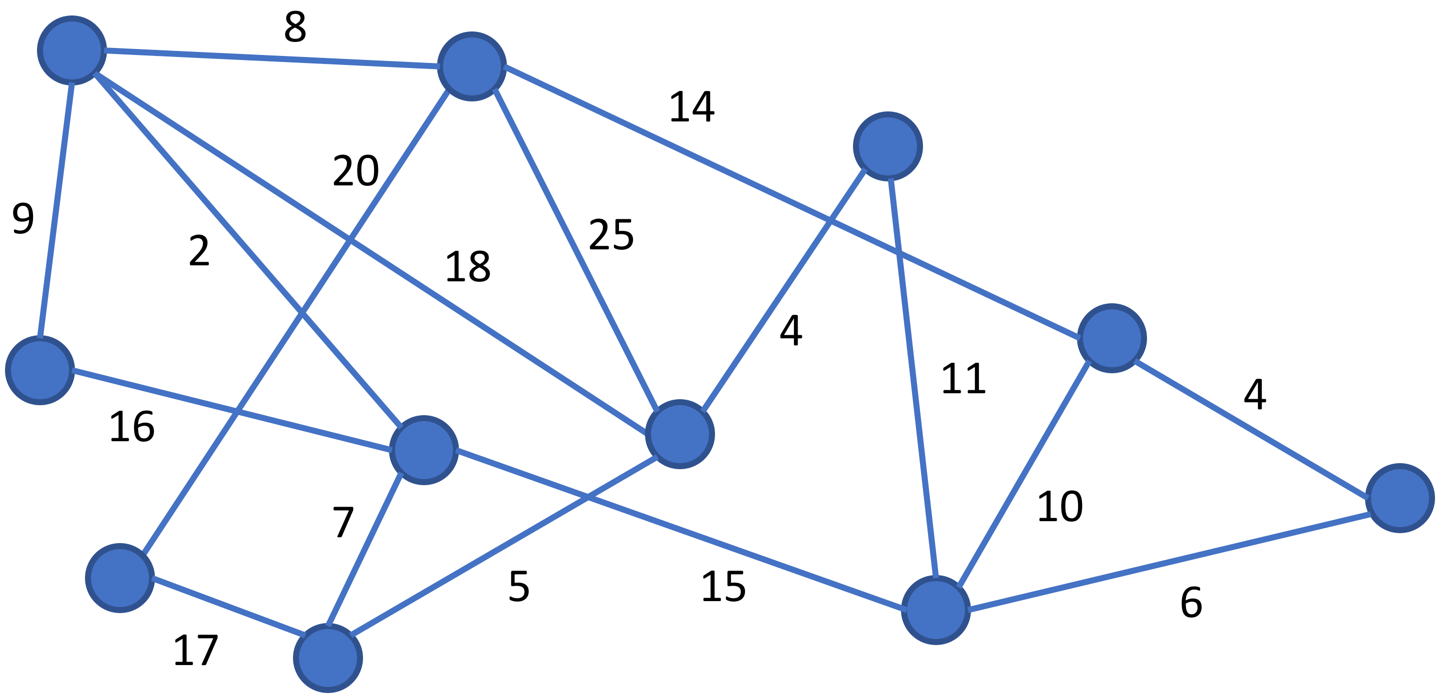
- Step2: Add an edge to the forest if the edge does not connect two vertices already in the same tree

Kruskal’s Algorithm

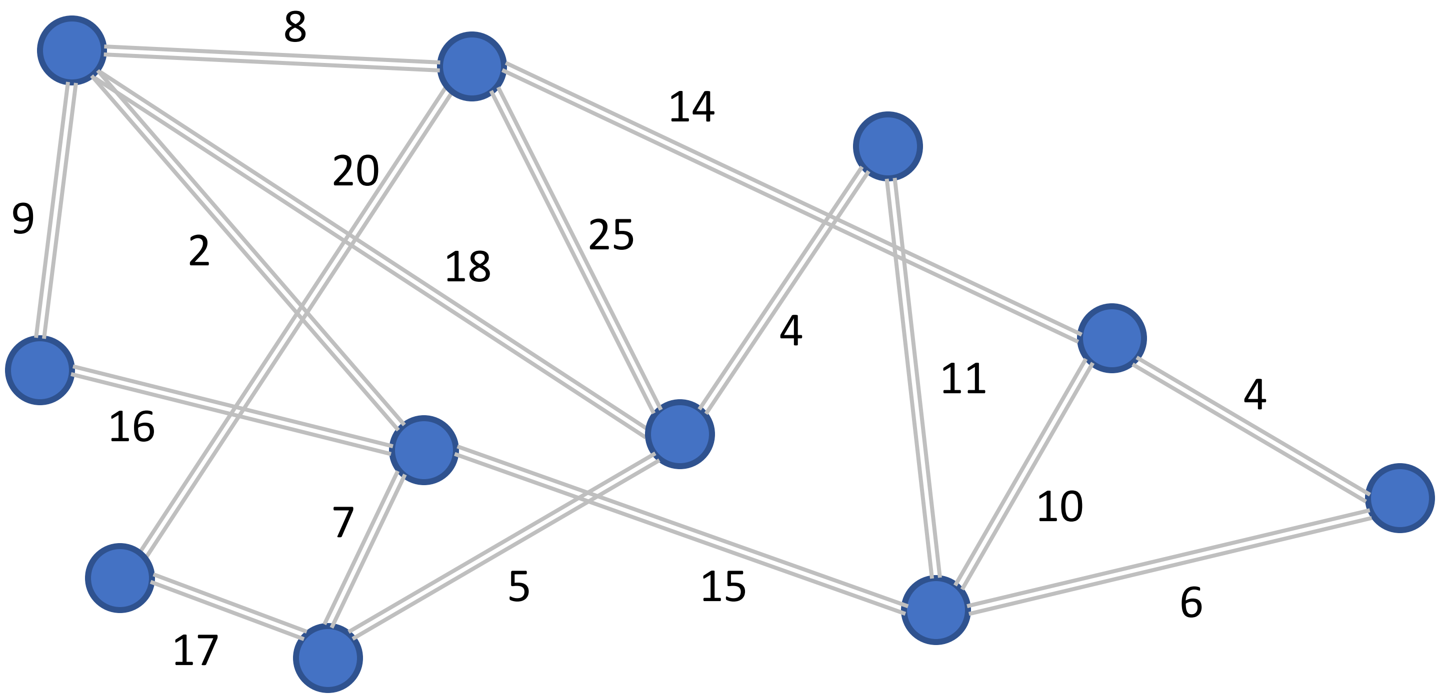
Algorithm 2 Kruskal’s algorithm.

function KRUSKAL(weighted digraph(G, c))
    disjoint sets ADT A
    initialize A with each vertex in its own set
    sort the edges in increasing order of cost
    for each edge $\{u, v\}$ in increasing cost order **do**
        if not $A.set(u) = A.set(v)$ **then**
            add this edge
            $A.union(A.set(u), A.set(v))$
return A

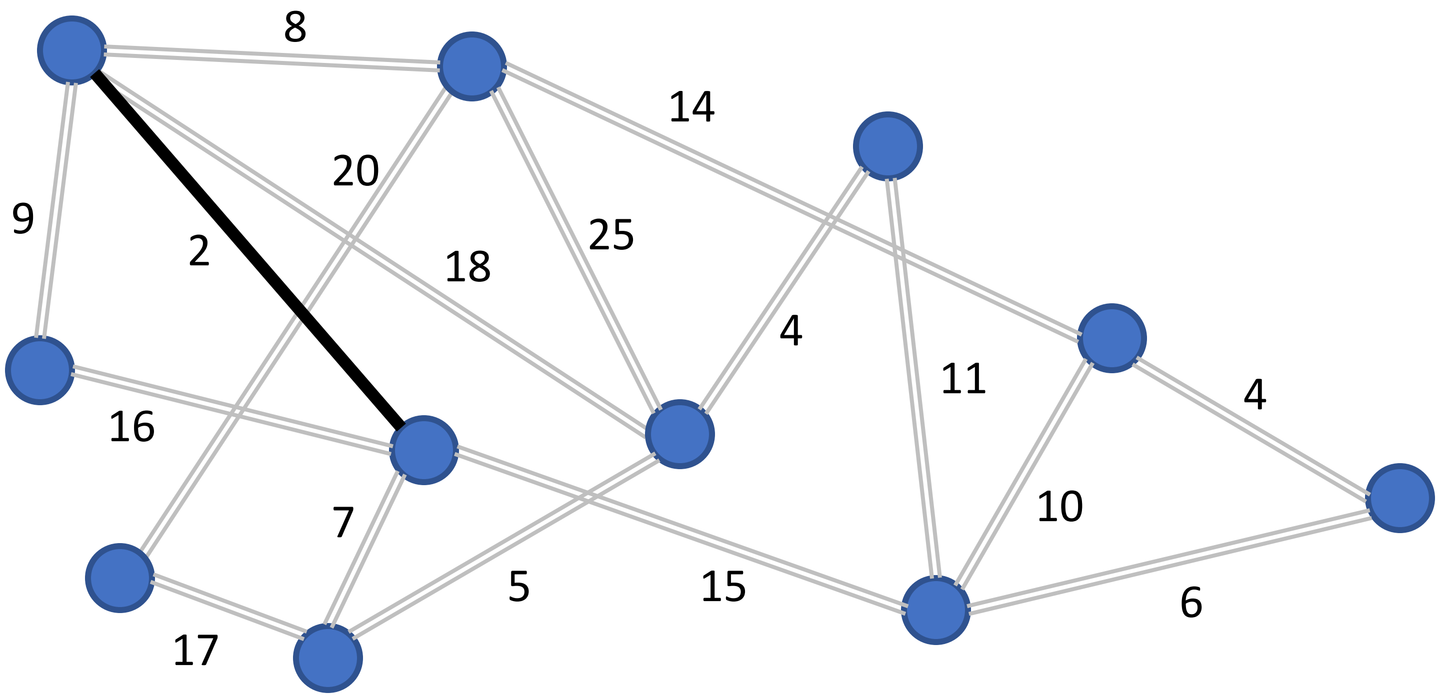
Example: Kruskal's Algorithm at Work



Example: Kruskal's Algorithm at Work

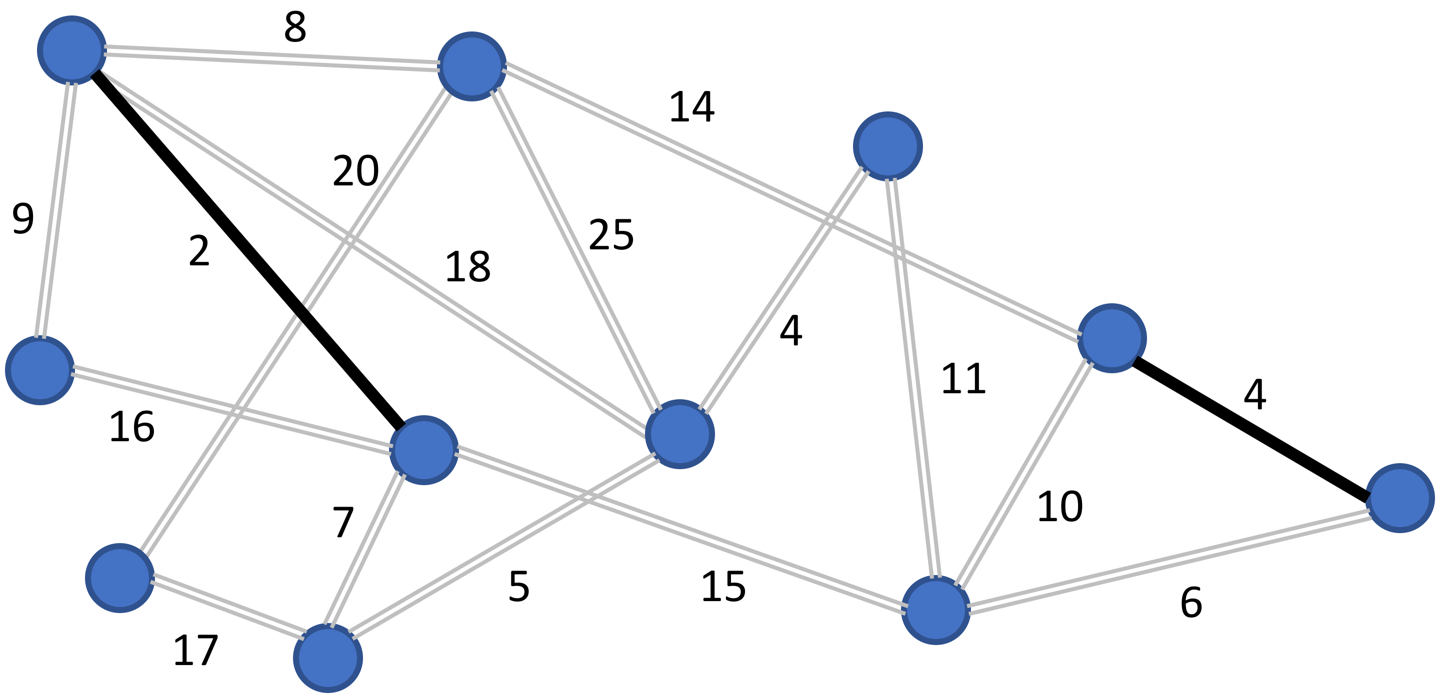


Example: Kruskal’s Algorithm at Work



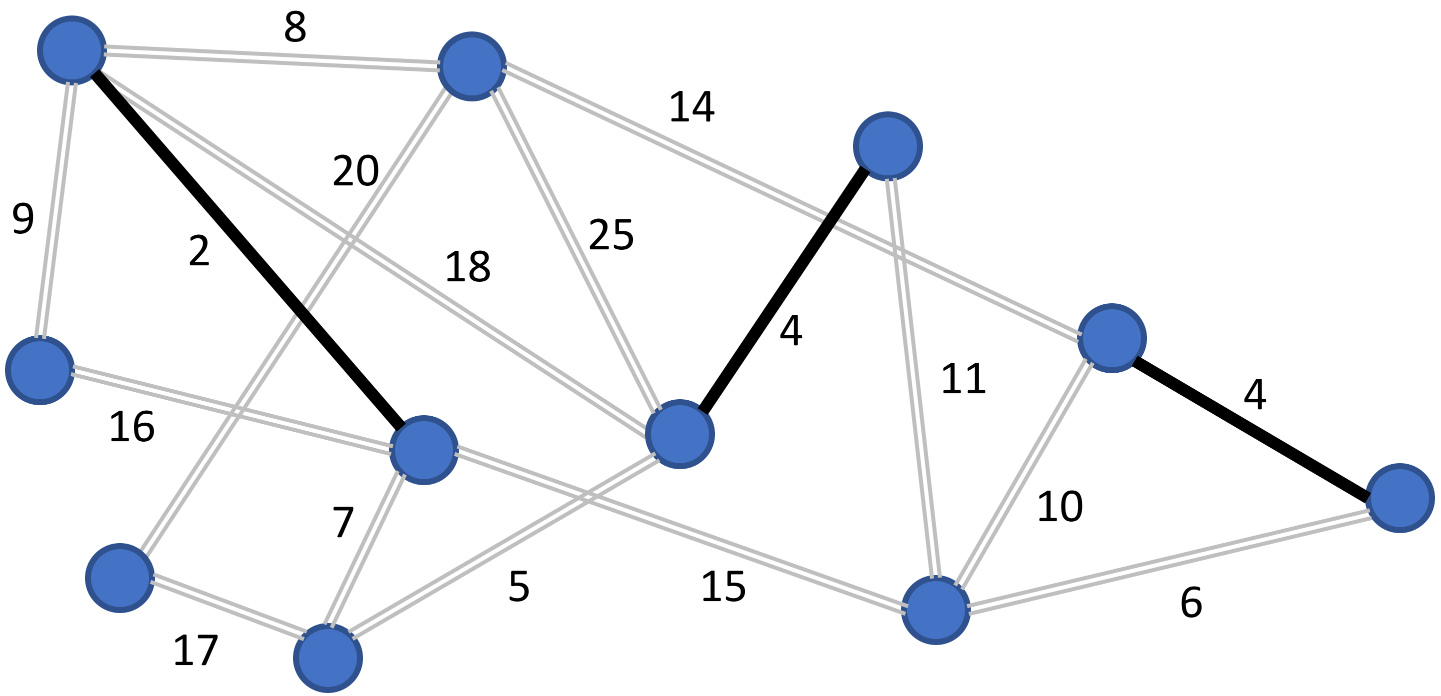
i=0

Example: Kruskal’s Algorithm at Work



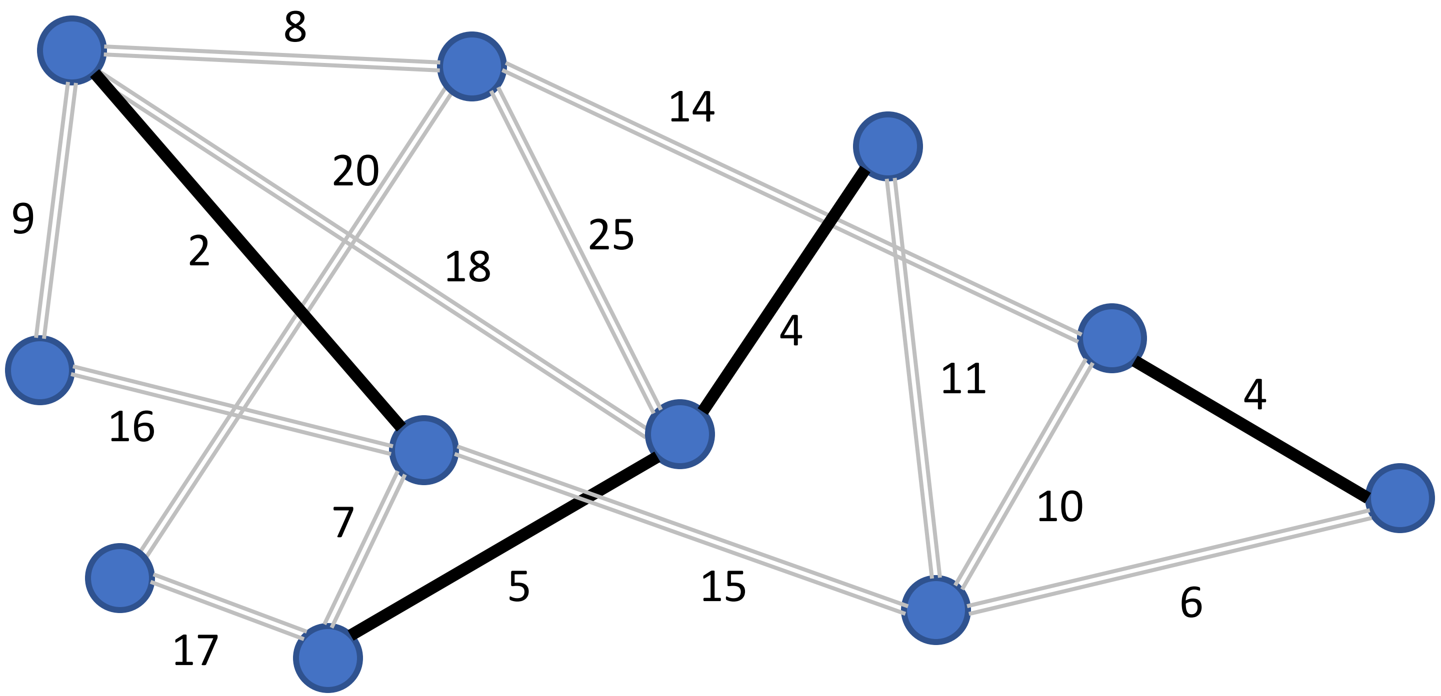
i=1

Example: Kruskal's Algorithm at Work



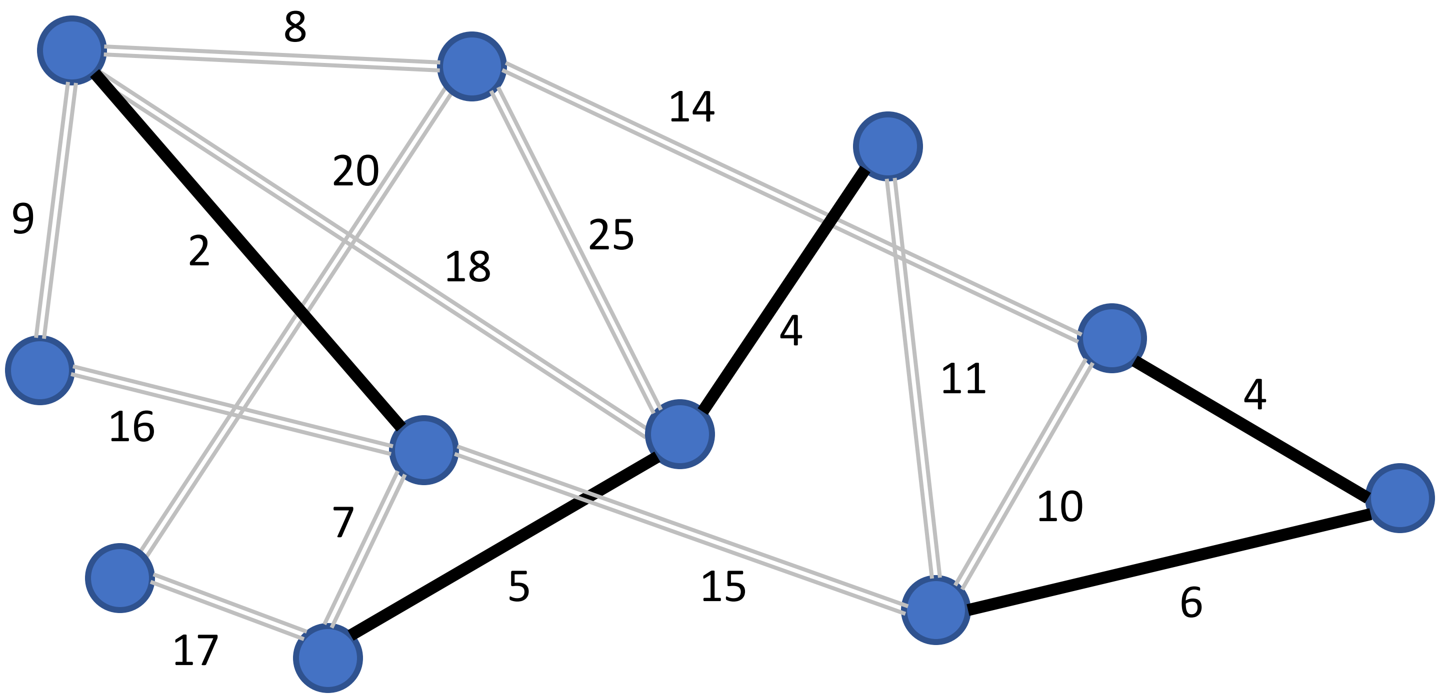
i=2

Example: Kruskal's Algorithm at Work



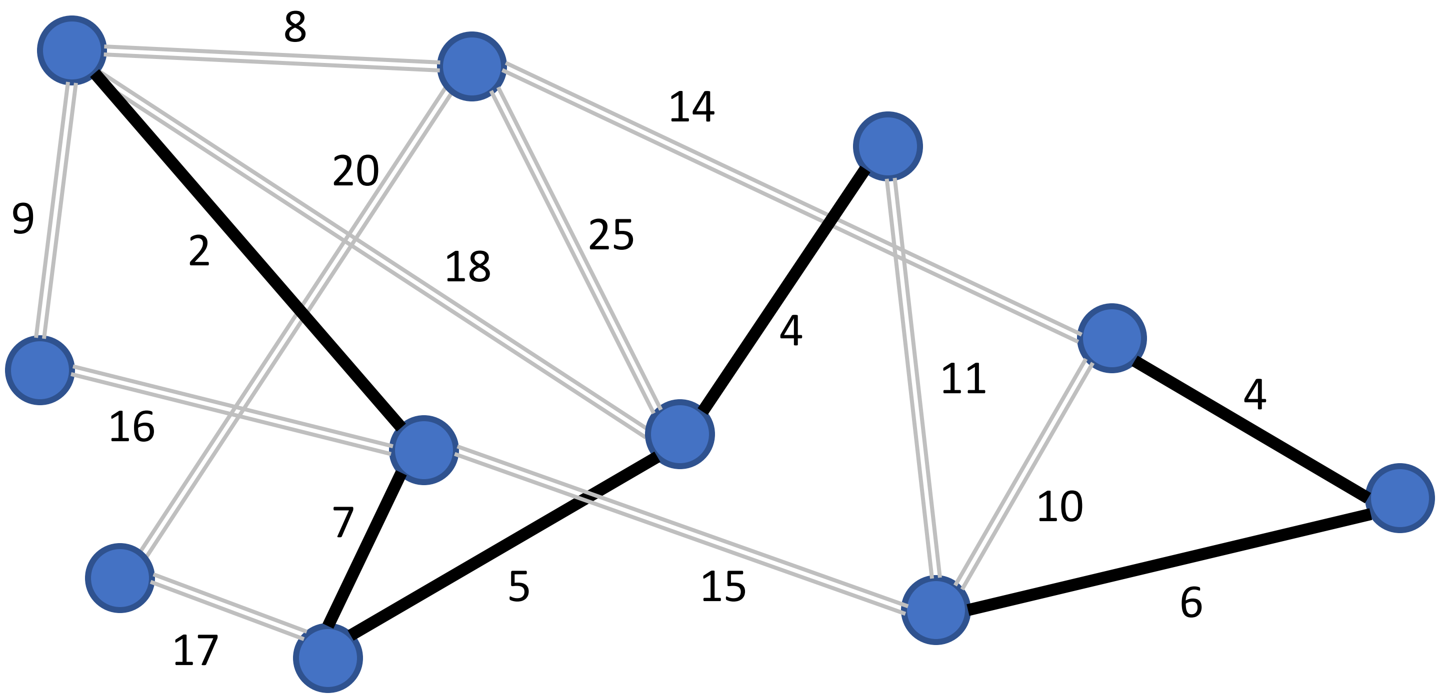
i=3

Example: Kruskal's Algorithm at Work



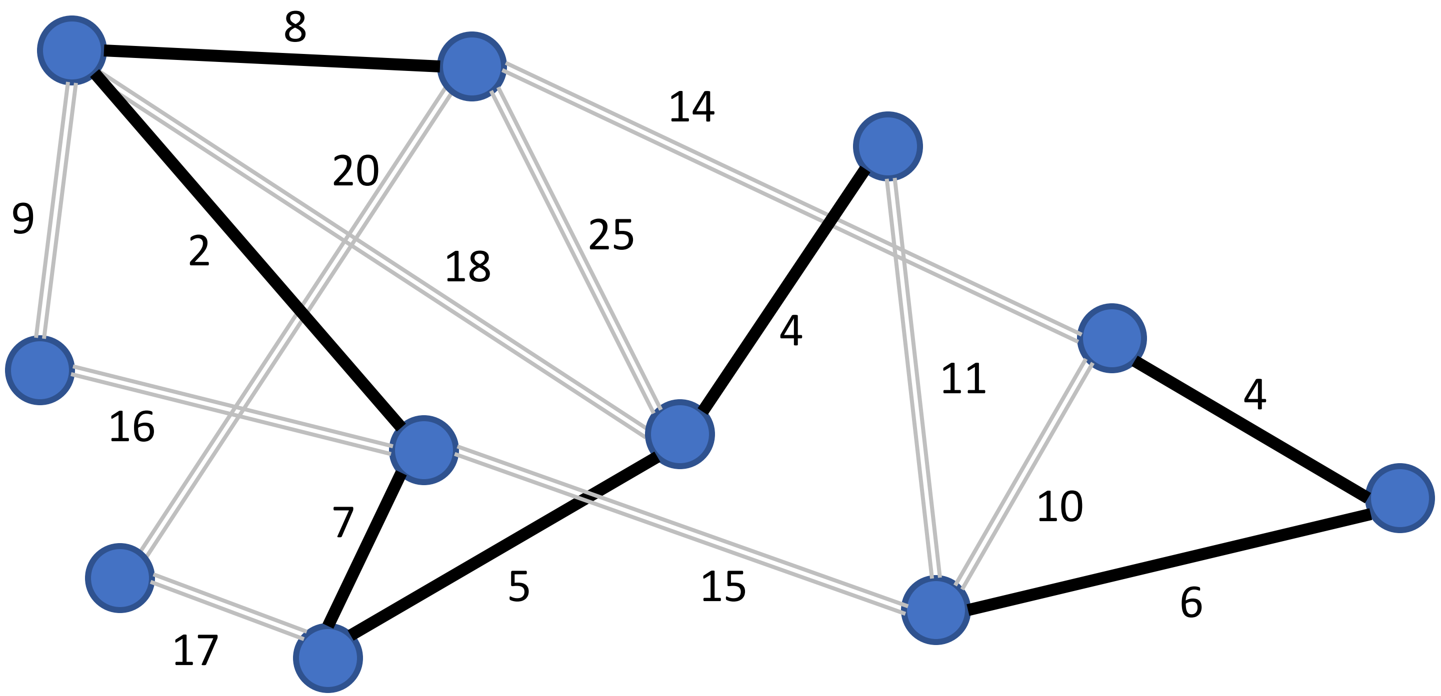
i=4

Example: Kruskal's Algorithm at Work



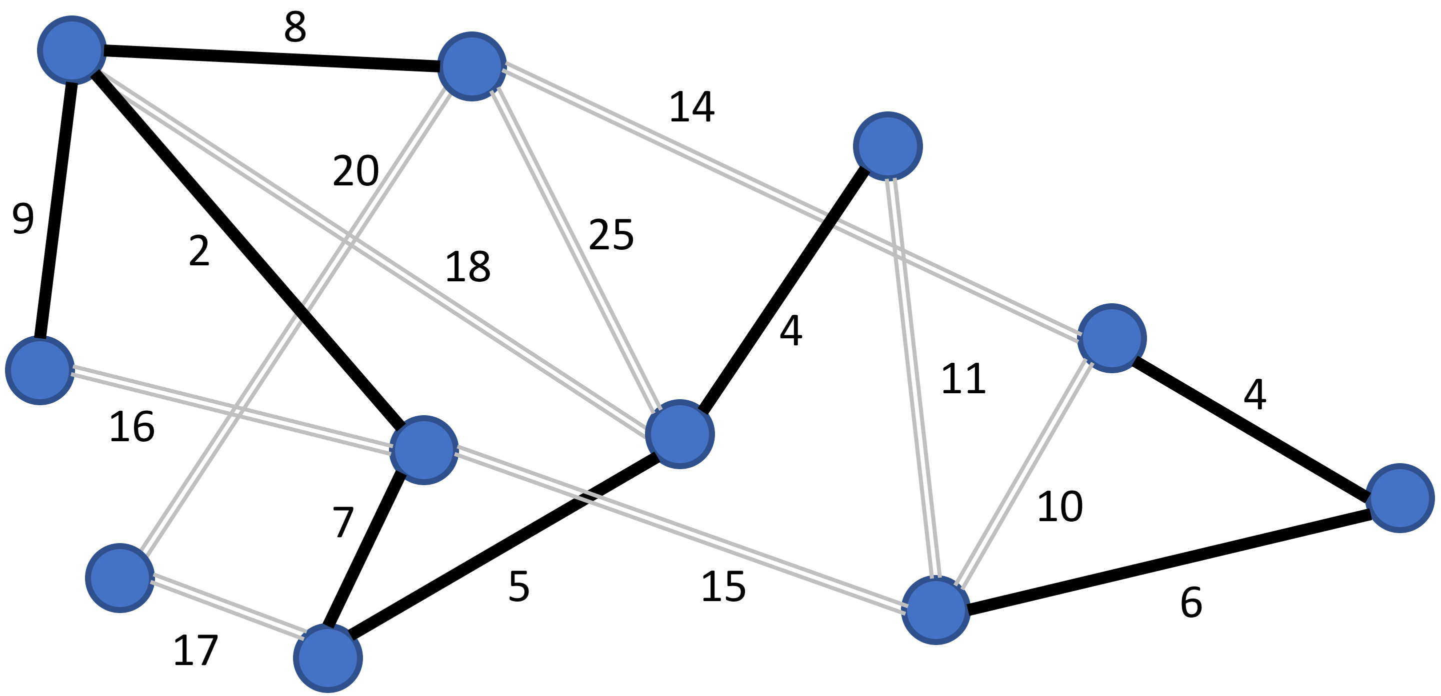
i=5

Example: Kruskal’s Algorithm at Work



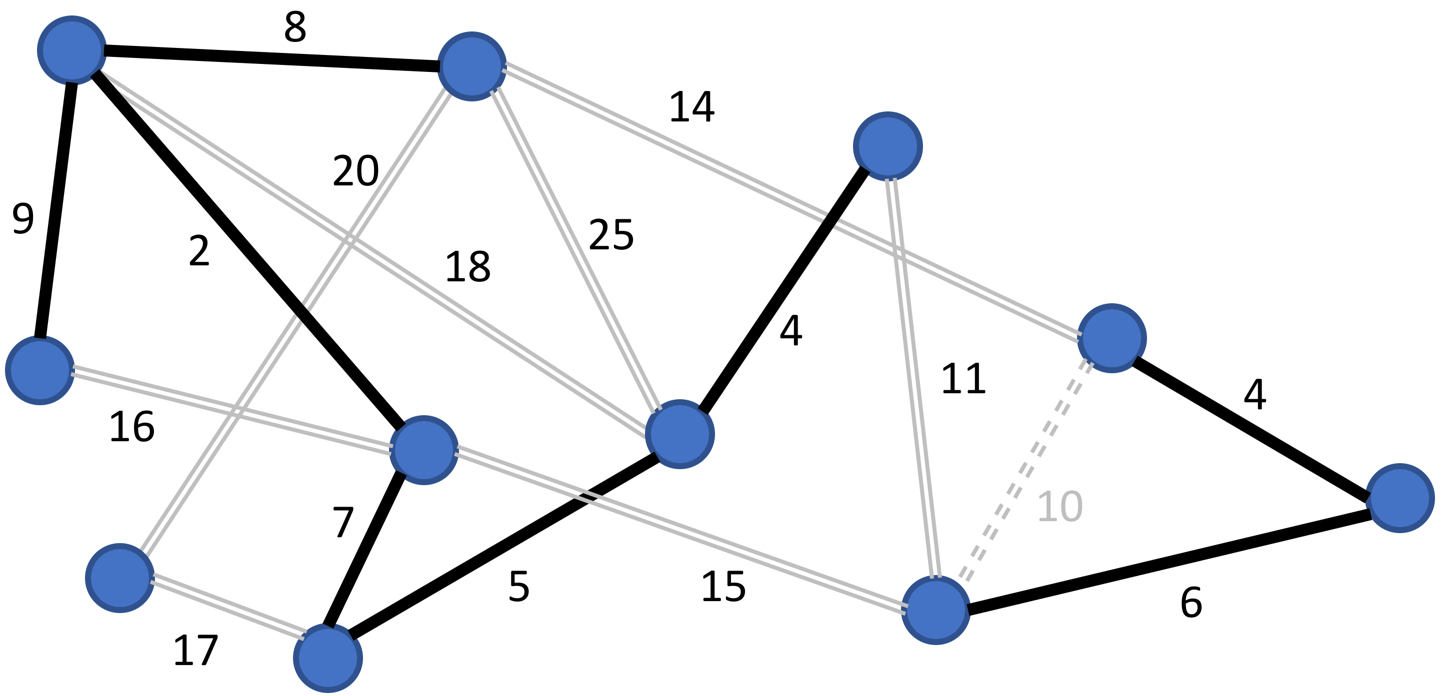
$i=6$

Example: Kruskal's Algorithm at Work



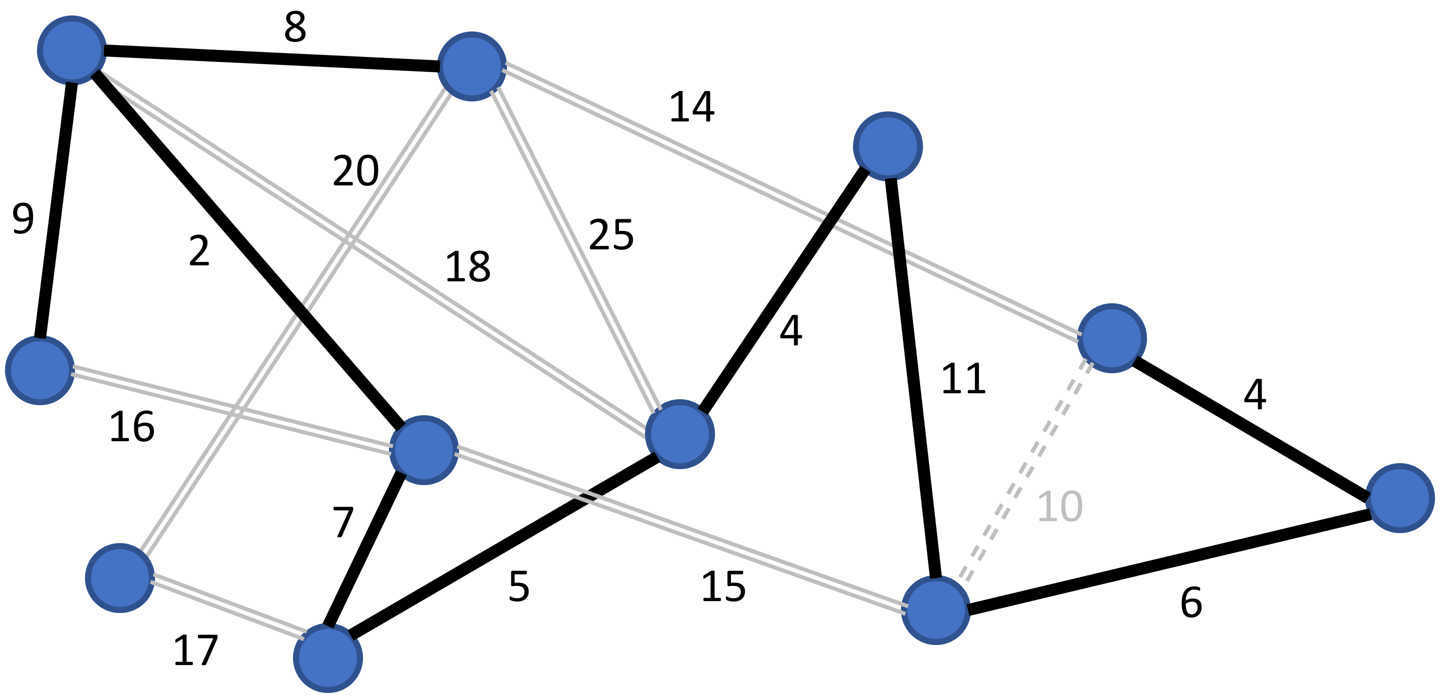
i=7

Example: Kruskal's Algorithm at Work



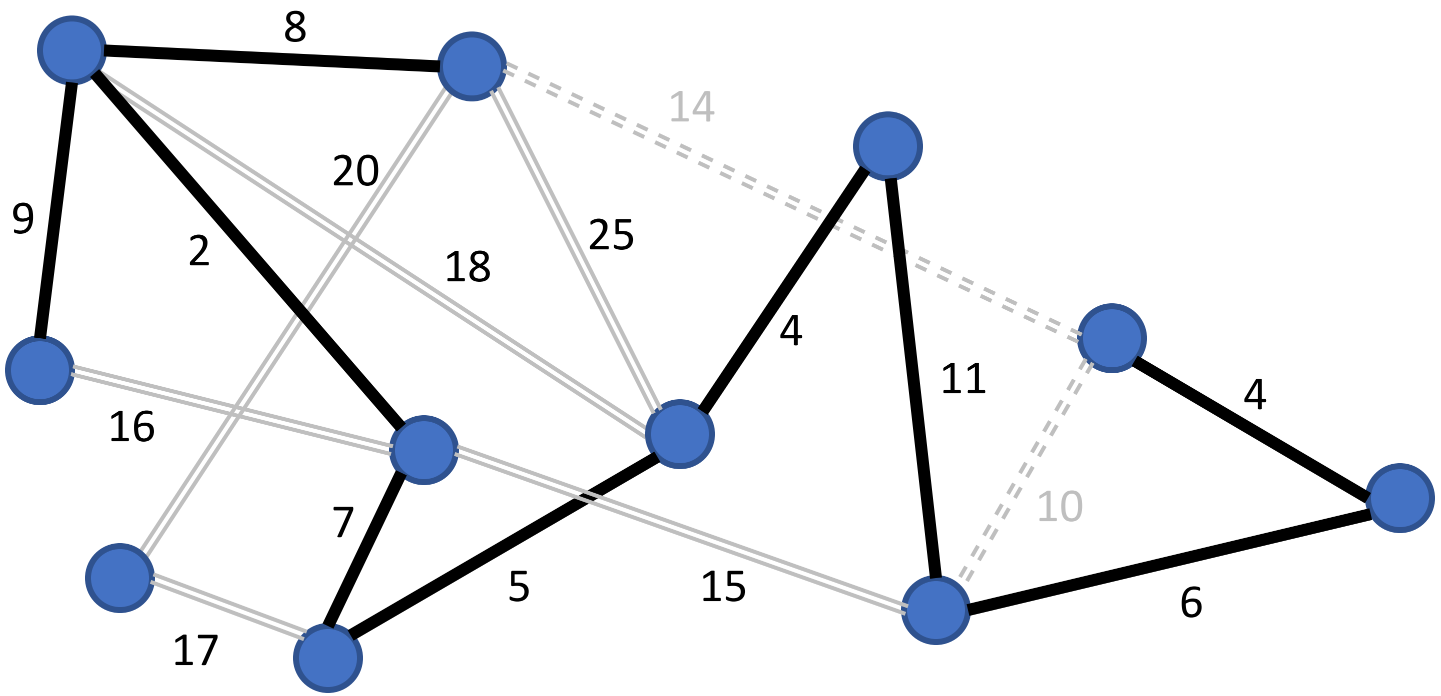
i=8

Example: Kruskal’s Algorithm at Work



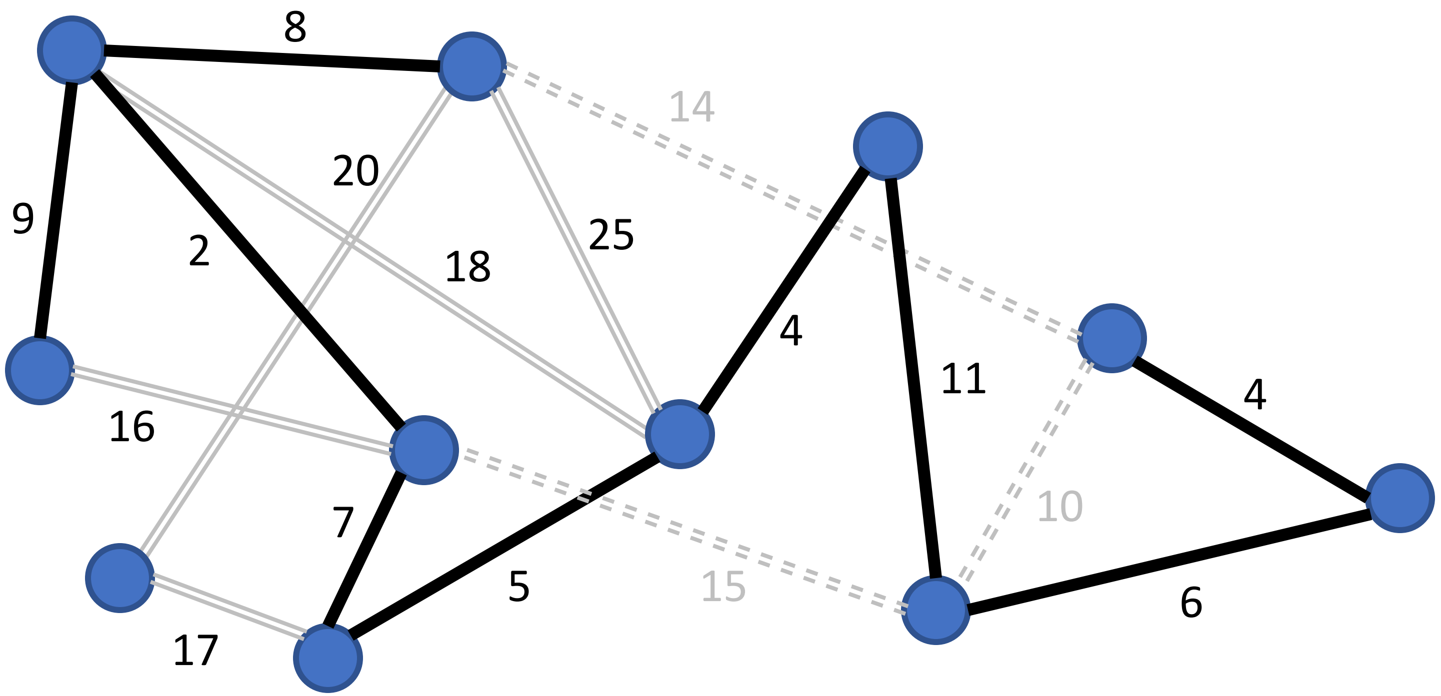
i=9

Example: Kruskal's Algorithm at Work



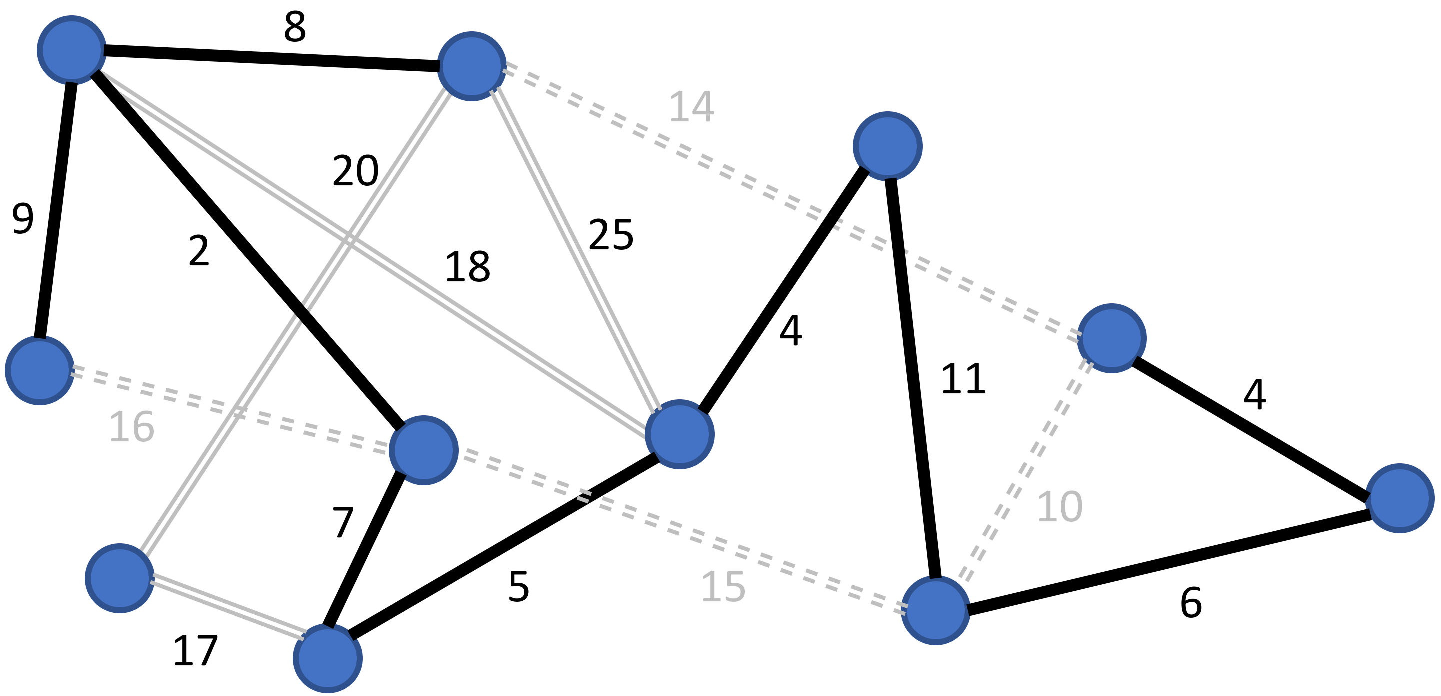
i=10

Example: Kruskal's Algorithm at Work



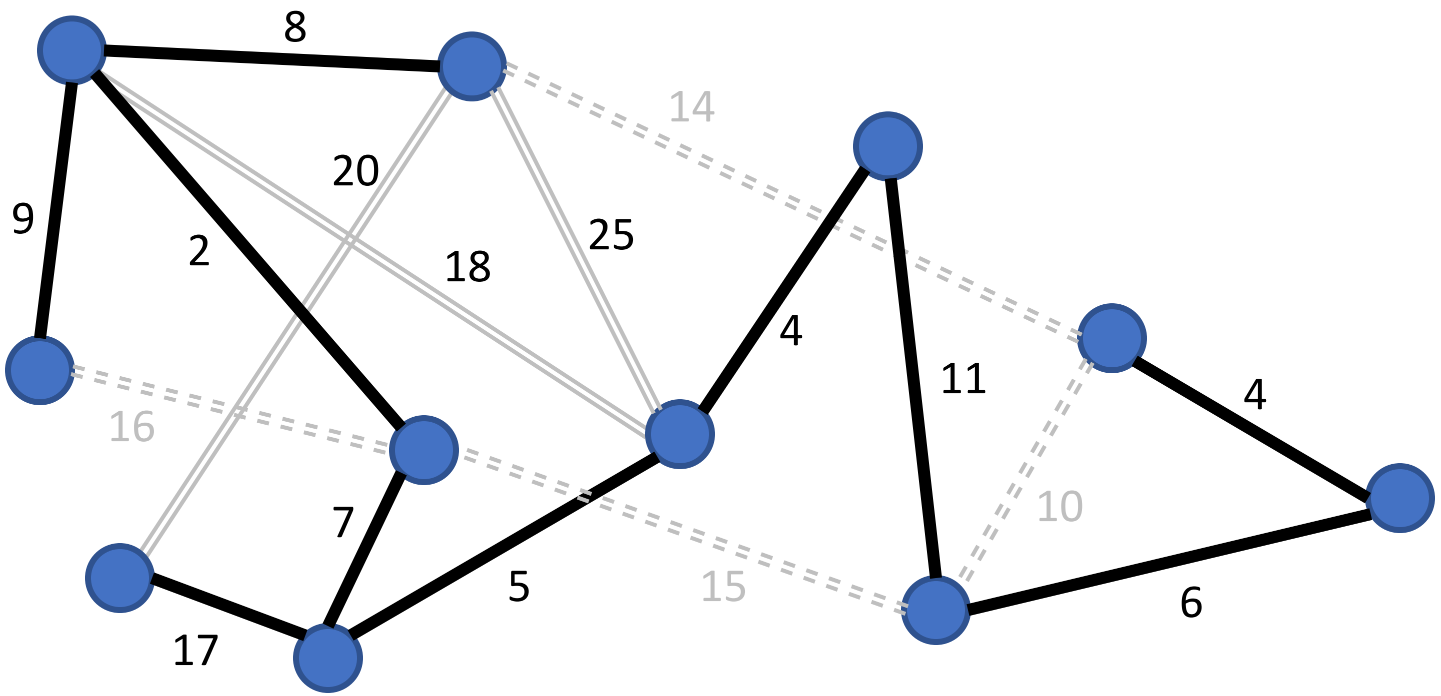
i=11

Example: Kruskal’s Algorithm at Work



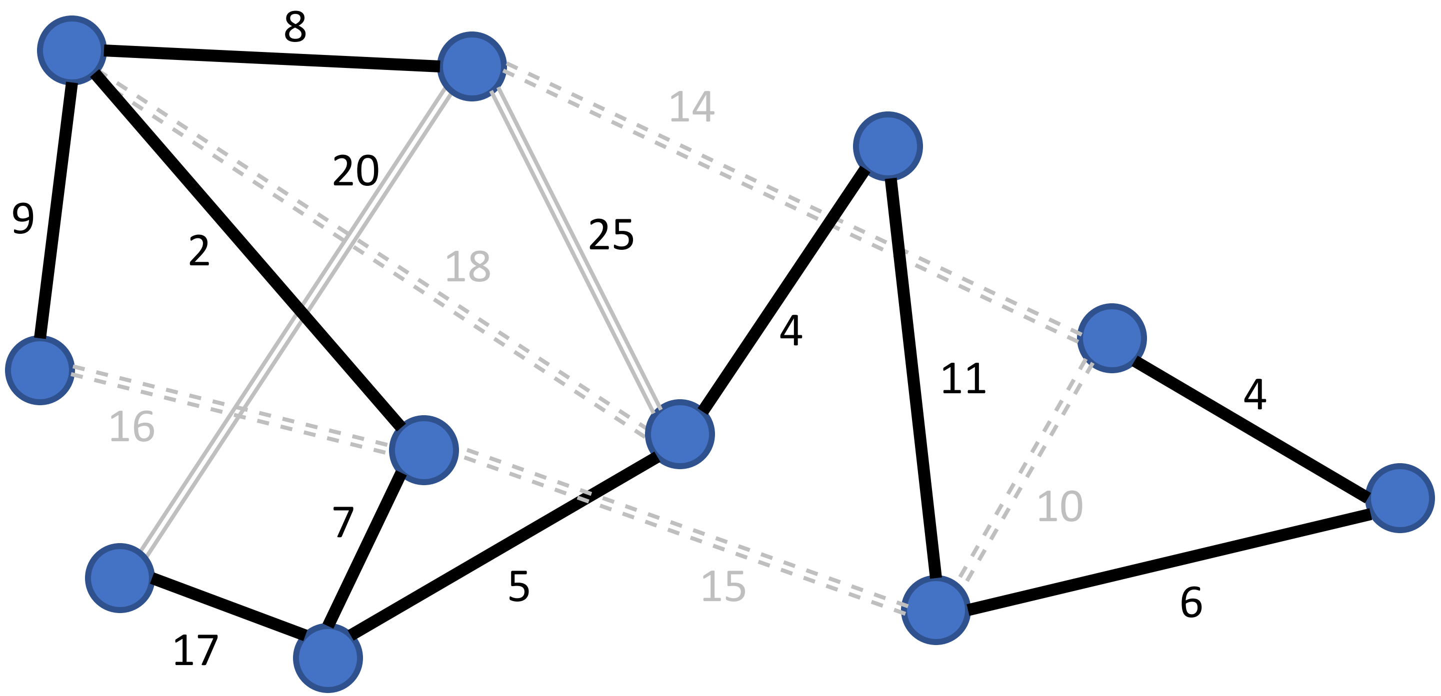
i=12

Example: Kruskal's Algorithm at Work



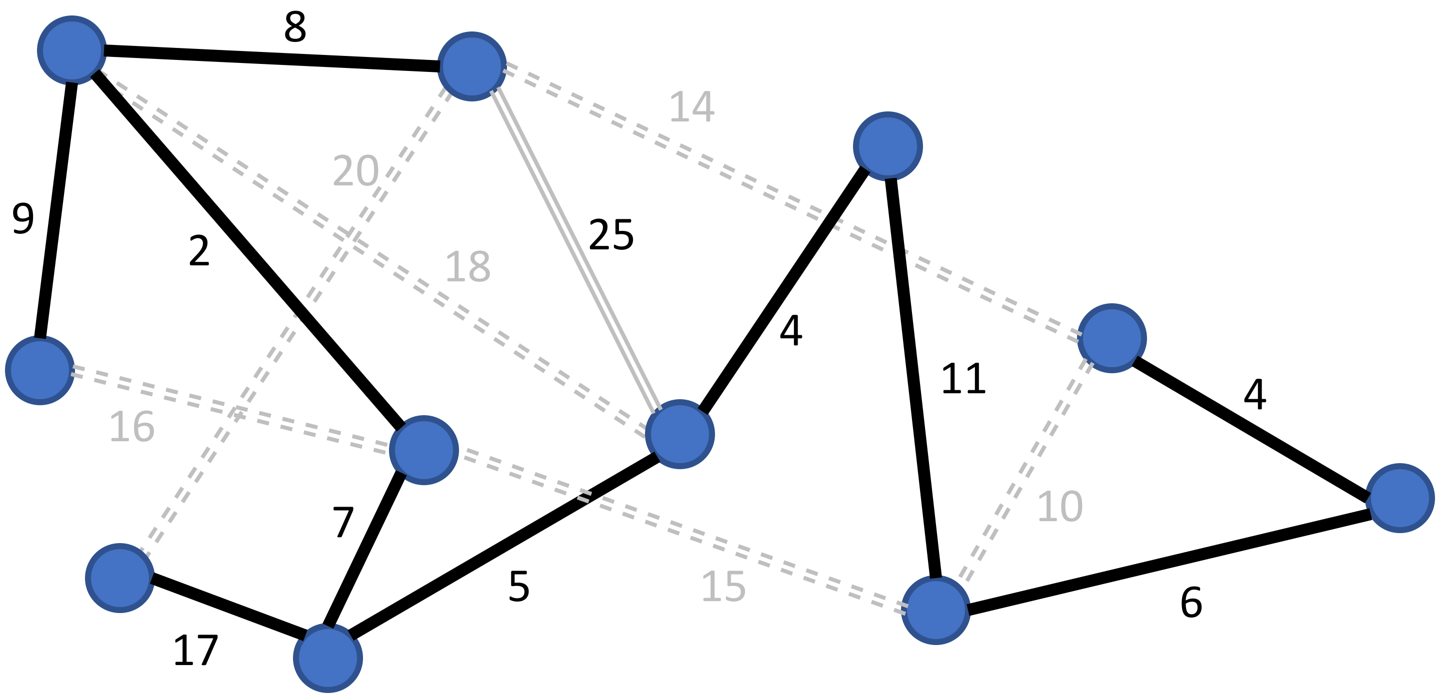
i=13

Example: Kruskal's Algorithm at Work



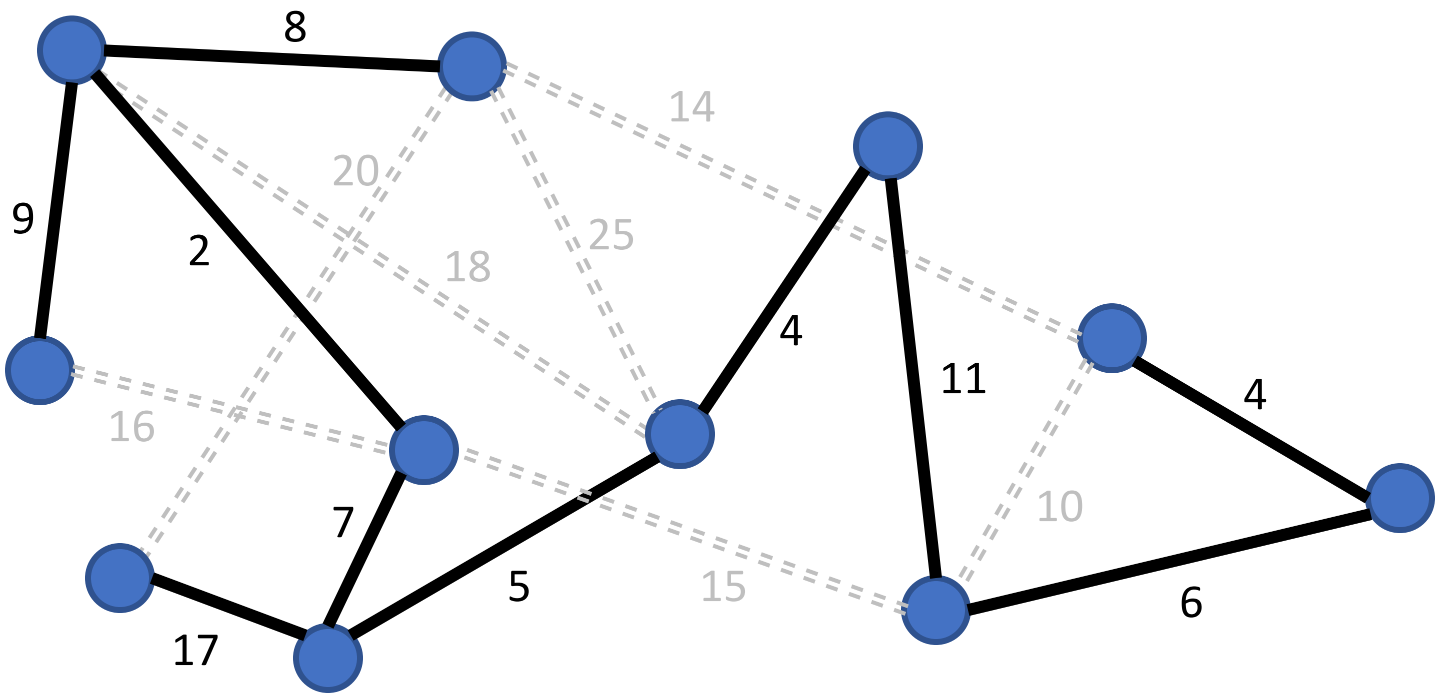
i=14

Example: Kruskal’s Algorithm at Work



i=15

Example: Kruskal's Algorithm at Work



i=16

Disjoint Set ADT

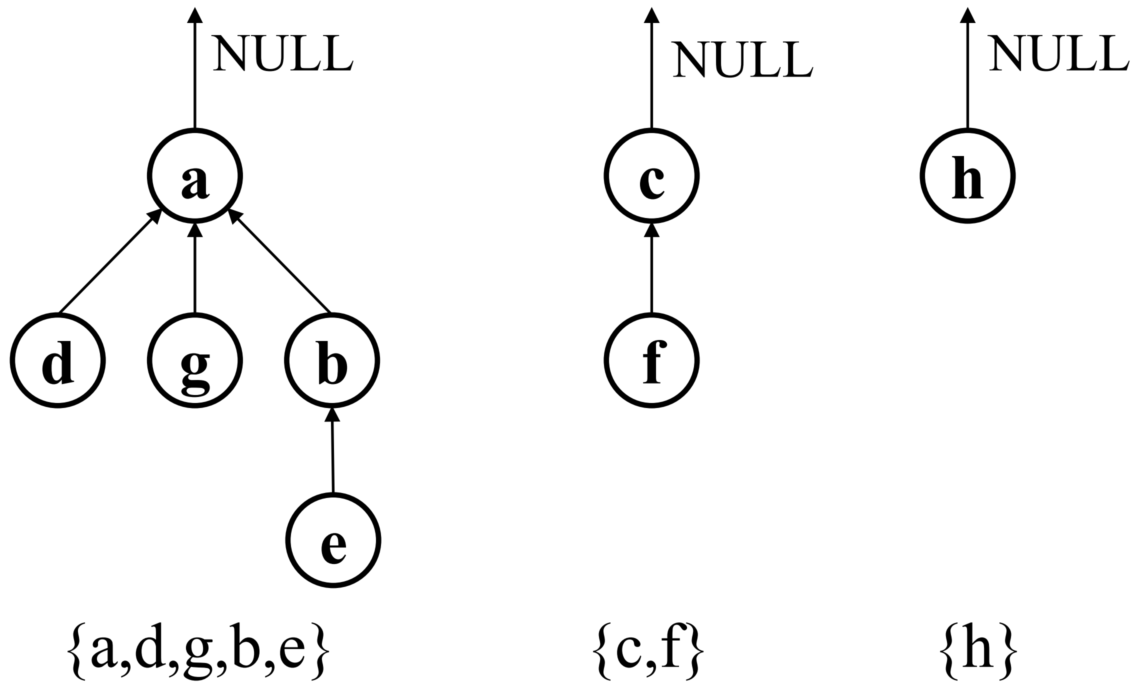
- **Disjoint Sets ADT**: Used to represent **a collection of sets containing objects that are related** to each other. Two standard operations:
  - Union operation - merges two sets by names (denoted by one of the objects in the set).
  - Find operation – given an object, returns the set (with its name) the object belongs to

Example:
Initial Sets = {1,4,8}, {2,3}, {6}, {7}, {5,9,10}

Find(4) returns a set named 1
Union(2, 6) results in a new set {2,3,6} with name 2.

Up-Tree Data Structure

- Each disjoint set is an up-tree with its root as its representative member
- All members of a given set are nodes in that set's up-tree



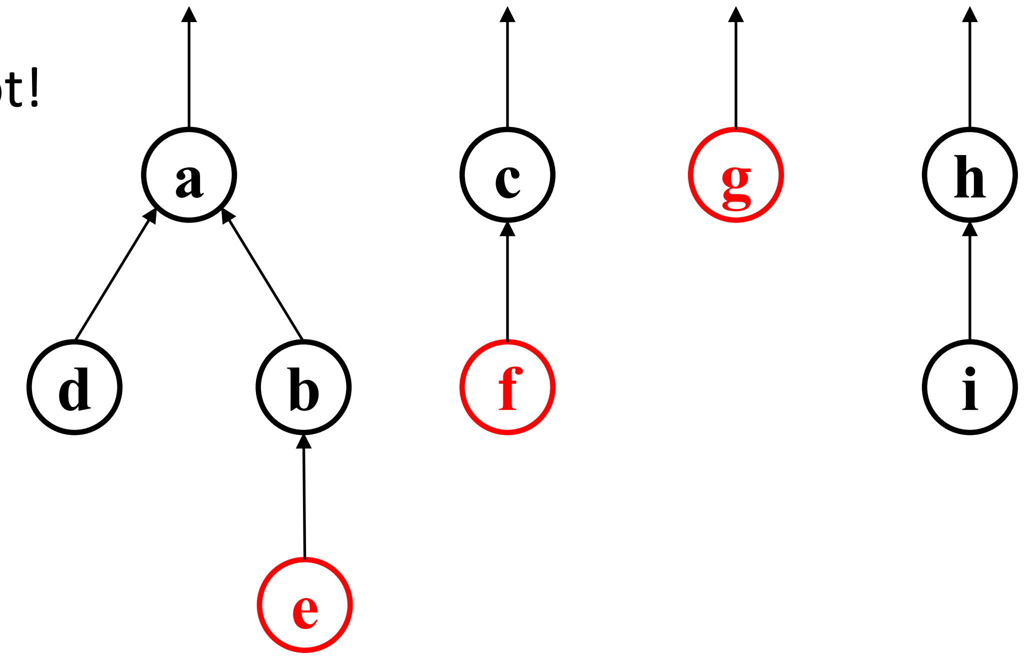
Example: Find

- Find: Just follow parent pointers to the root!

find(f) = c

find(e) = a

find(g) = g



Runtime depends on tree depth

-(Tree size)

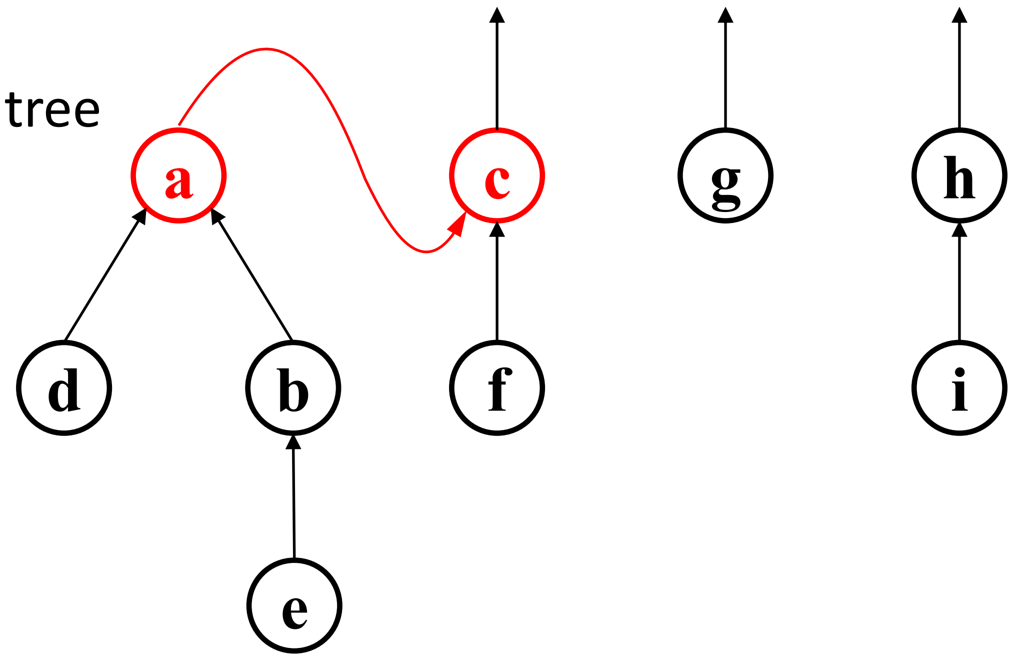
Array up:

0 (a)	1 (b)	2 (c)	3 (d)	4 (e)	5 (f)	6 (g)	7 (h)	8 (i)
-4	0	-2	0	1	2	-1	-2	7

Example: Union

- Union: put root under the root of the other tree

union(c,a)



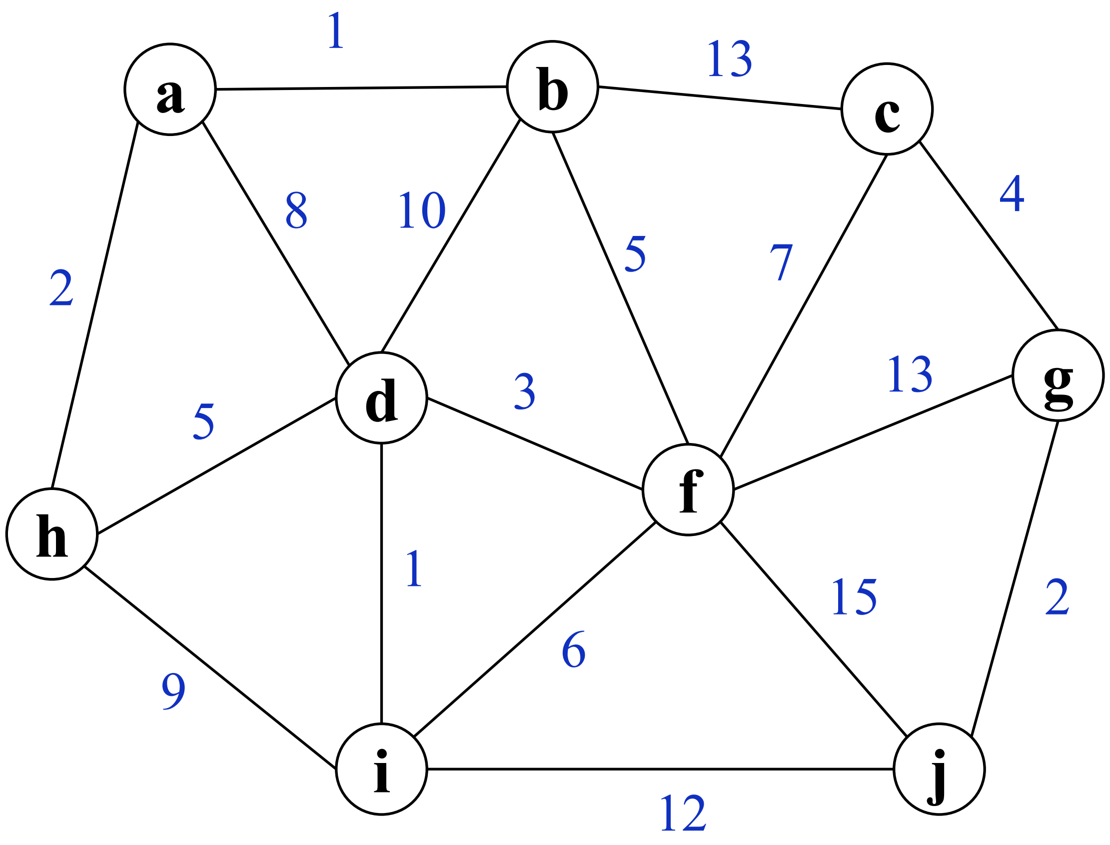
Runtime = O(1)

Array up:

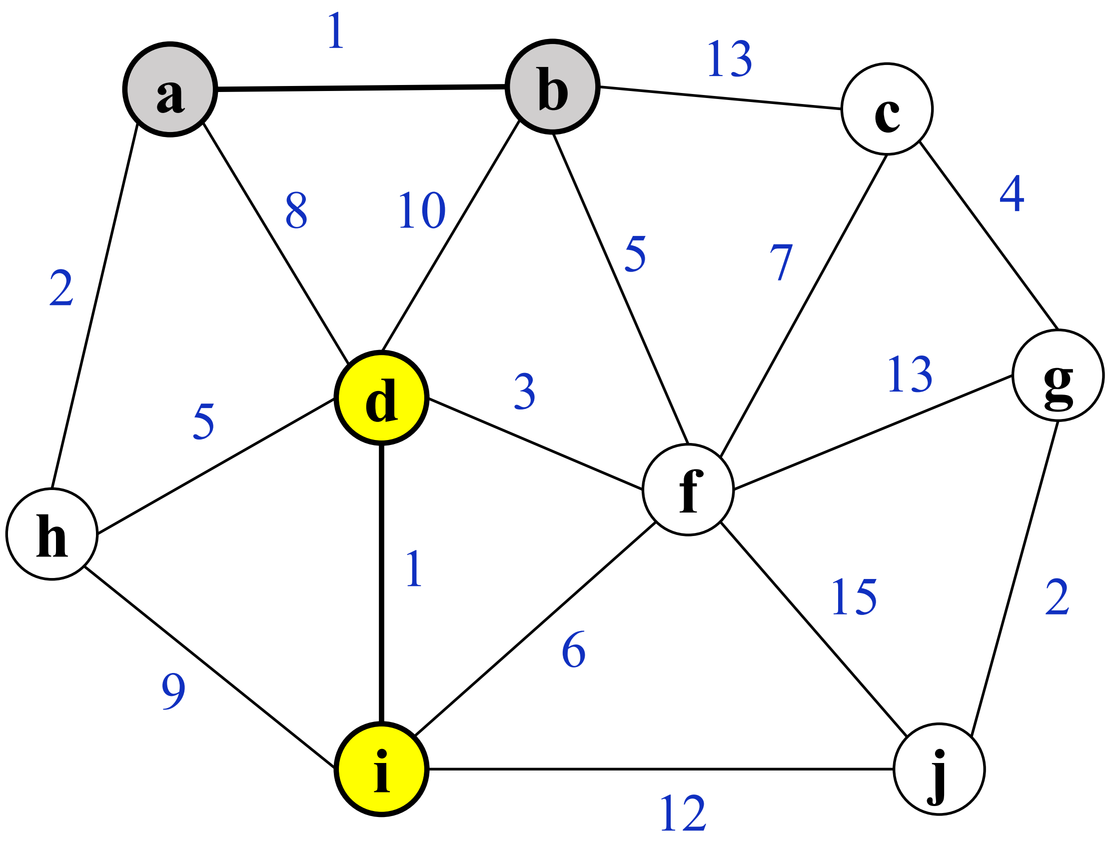
0 (a)	1 (b)	2 (c)	3 (d)	4 (e)	5 (f)	6 (g)	7 (h)	8 (i)
2	0	-6	0	1	2	-1	-2	7

Change a (from -4) to c(= 2), update size

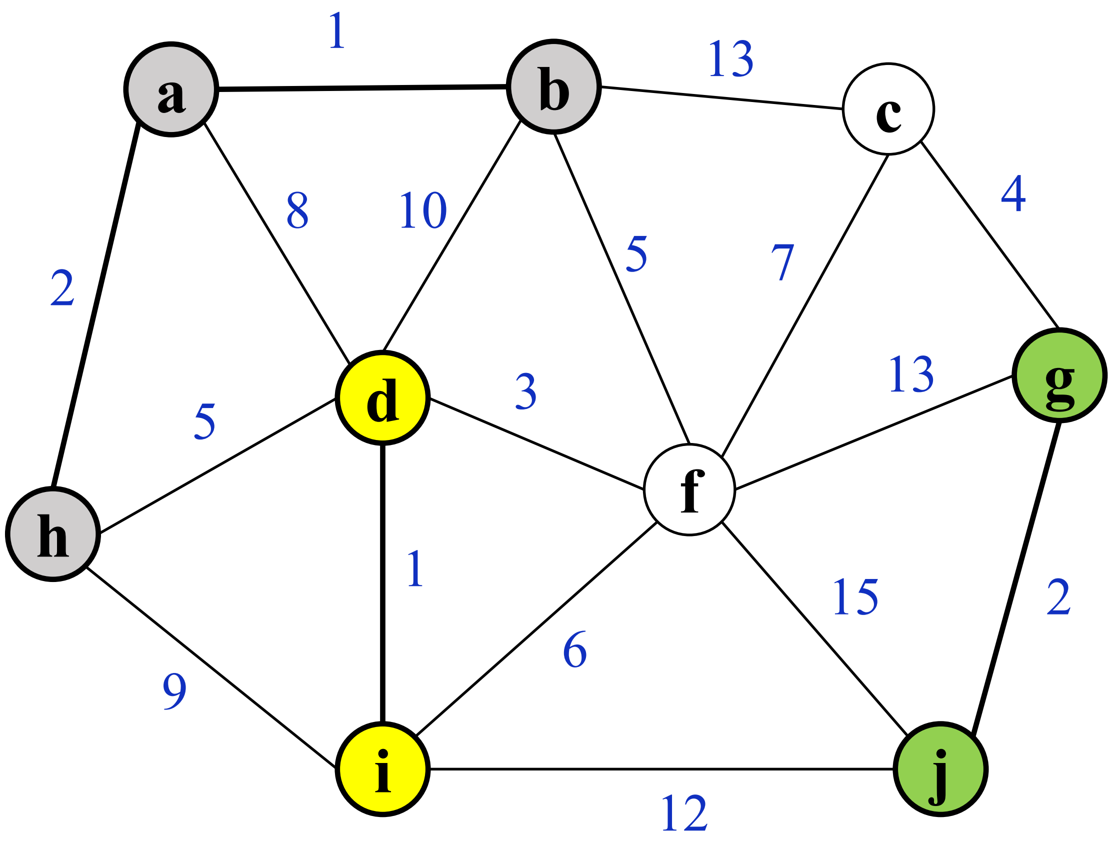
Illustrating Kruskal's algorithm



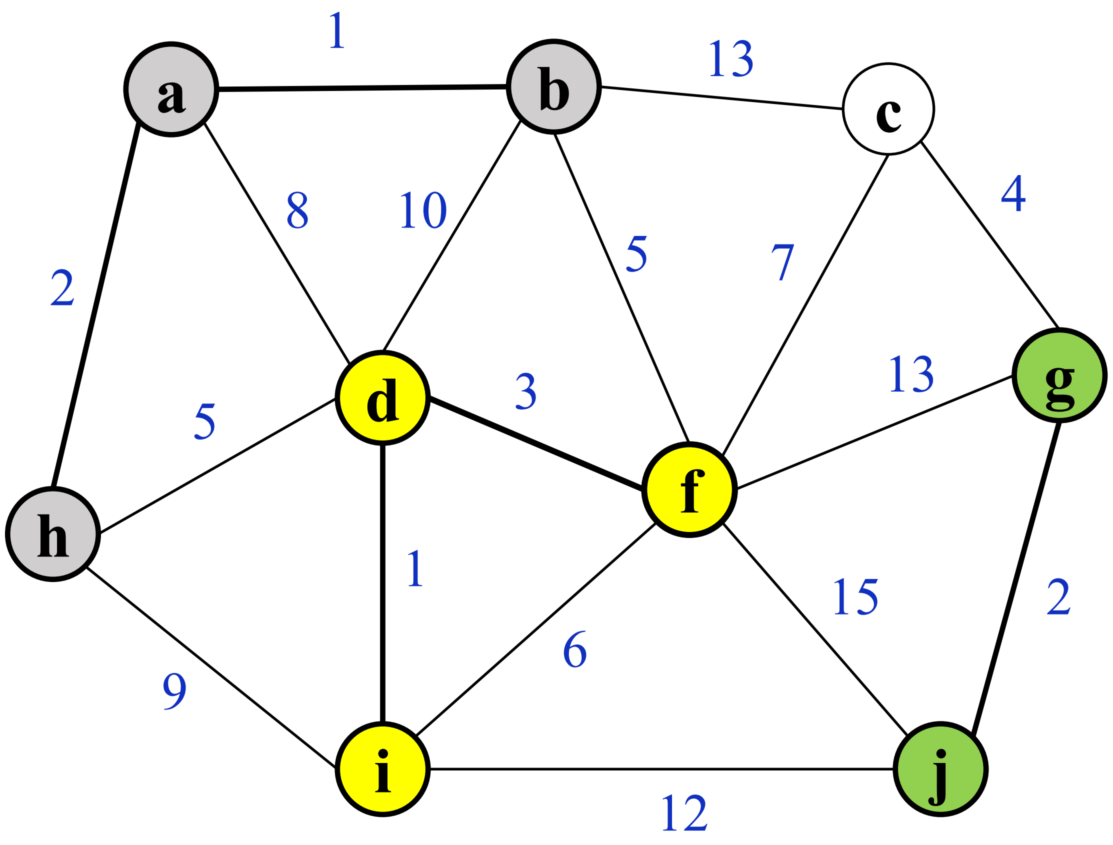
Illustrating Kruskal's algorithm



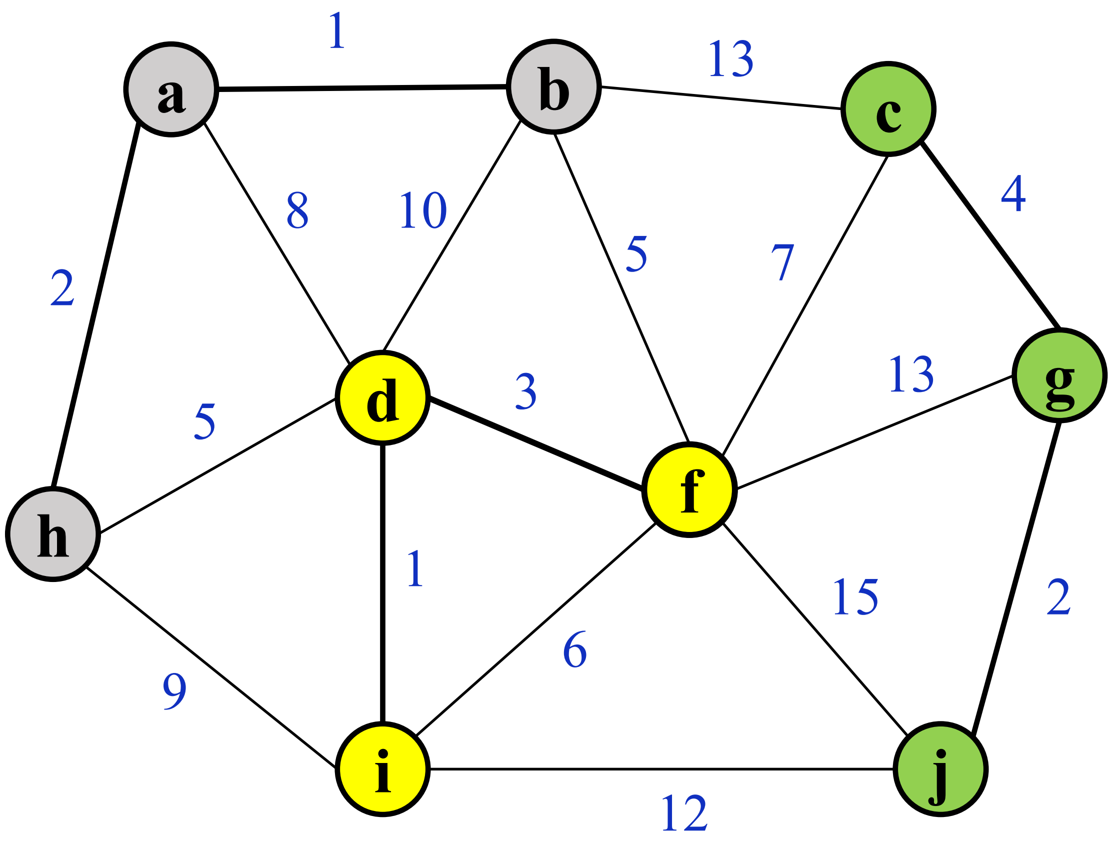
Illustrating Kruskal’s algorithm



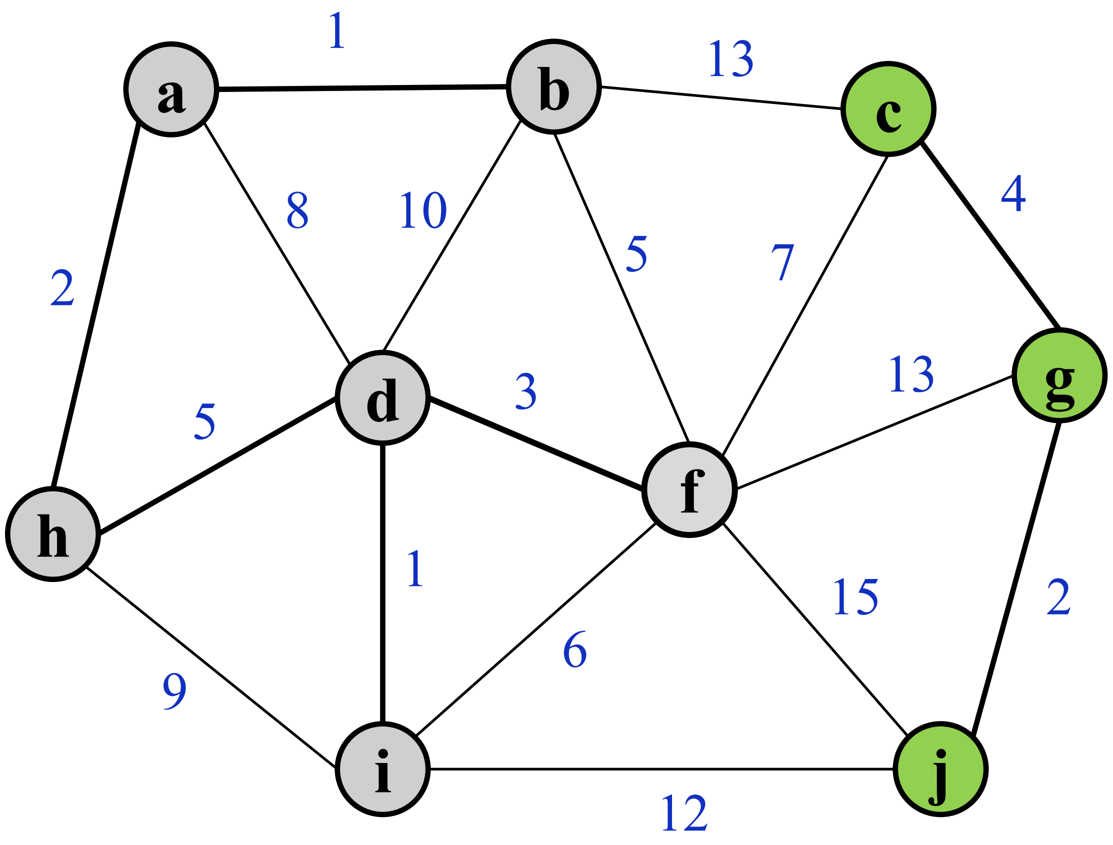
Illustrating Kruskal's algorithm



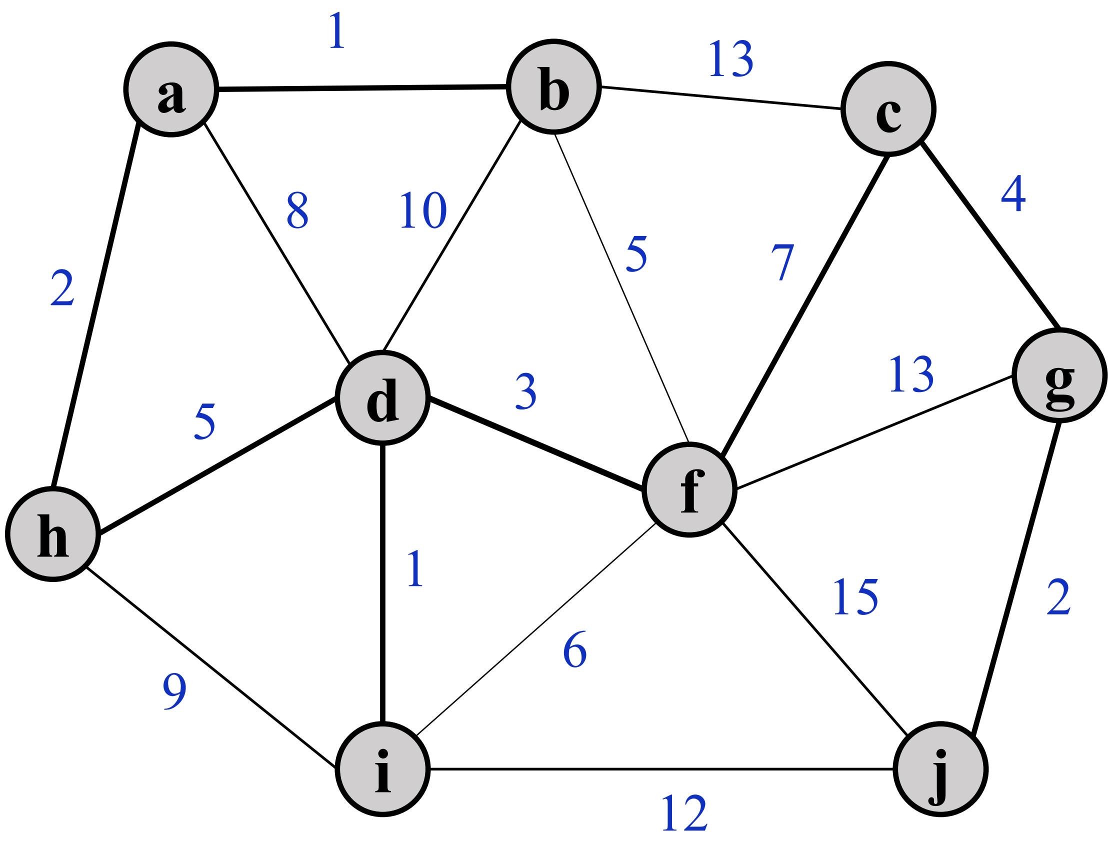
Illustrating Kruskal's algorithm



Illustrating Kruskal's algorithm



Illustrating Kruskal’s algorithm



Time Complexity Analysis

- Time complexity is dominated by having to sort the edges by weight: $O(e \log e)$, the time to sort.
  - Note e is at most n^2 and log e = log n^2 = 2 log n.
  - Therefore $O(e \log n)$

- Kruskal's algorithm is a little harder to program but easier to do by hand

Minimum Spanning Tree (MST) Summary

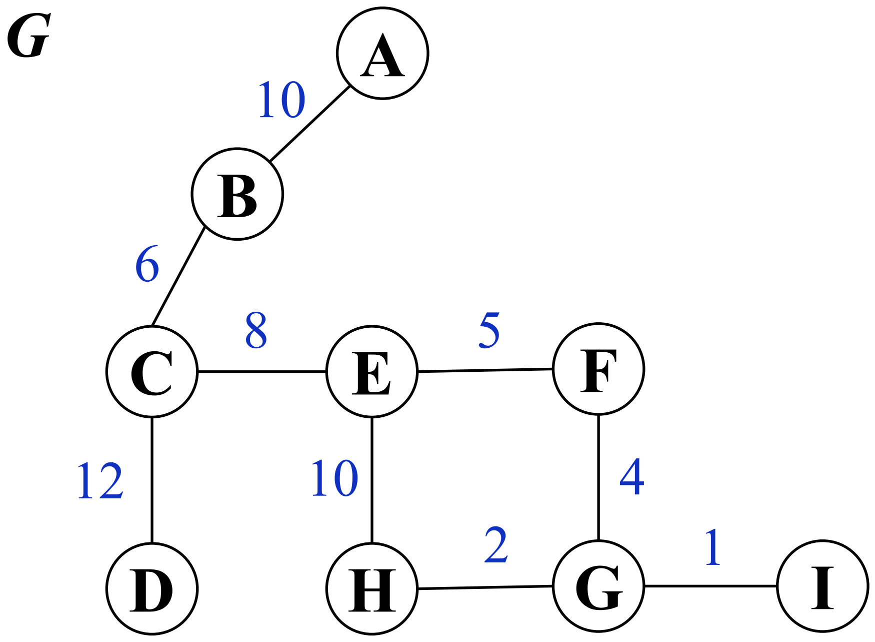
- **Fact.**

1. The most expensive edge, if unique, of a cycle in an edge-weighted graph G is not in any MST. (Otherwise, at least one of those equally expensive edges of the cycle must not be in each MST.)

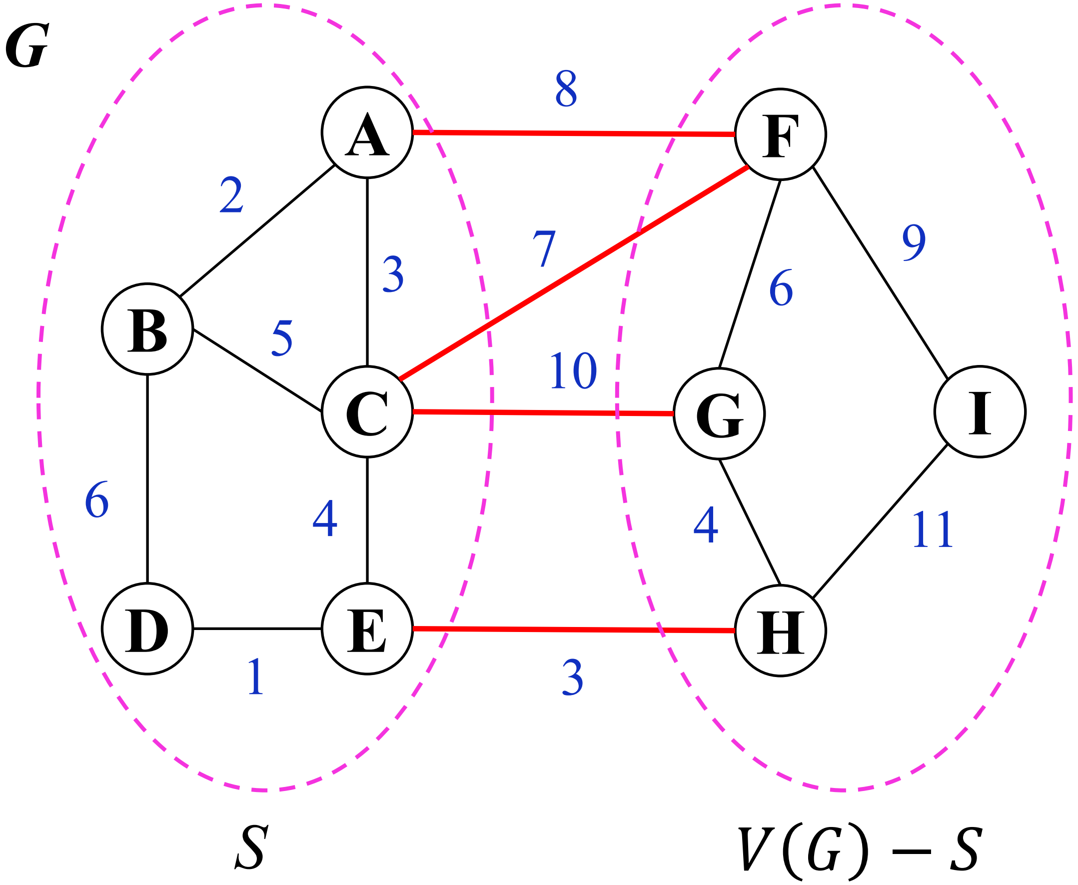
2. The minimum cost edge, if unique, between any non-empty strict subset S of $V(G)$ and the $V(G)\backslash$S is in the MST. (Otherwise, at least one of these minimum cost edges is in each MST.)

Max/Min Cost Edges in MST

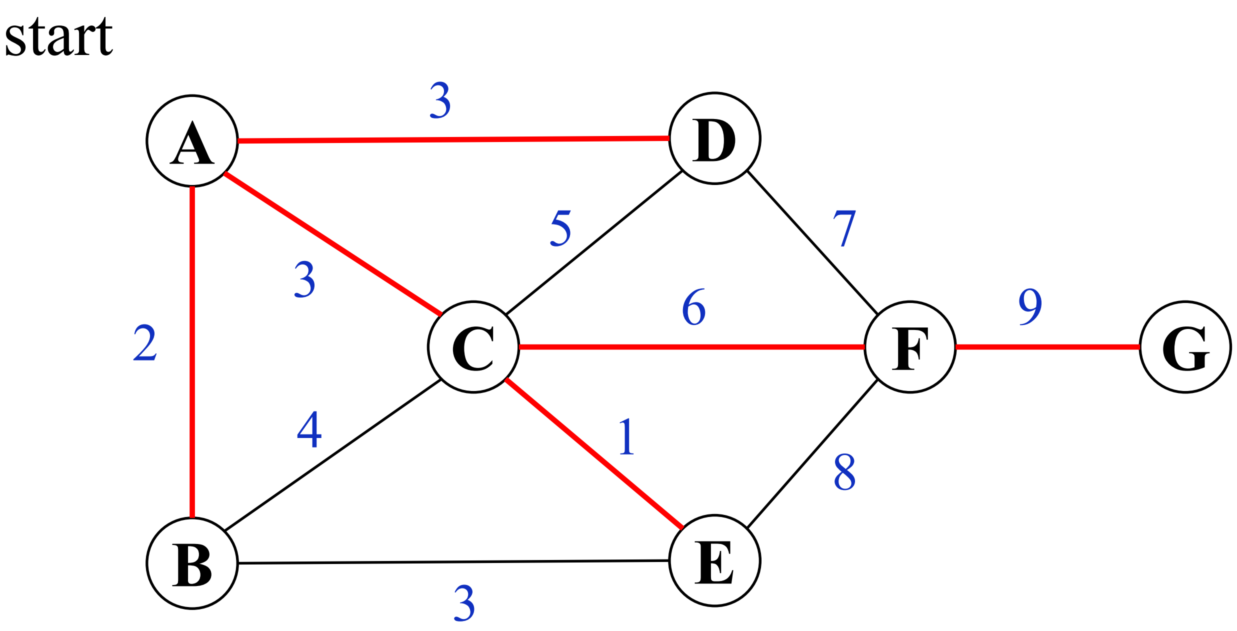
$S = \{A, B, C, D, E\}$
$V(G) - S = \{F, G, H, I\}$
MST of G contains $\{E, H\}$



MST does not contain $\{E, H\}$



Prim's Algorithm

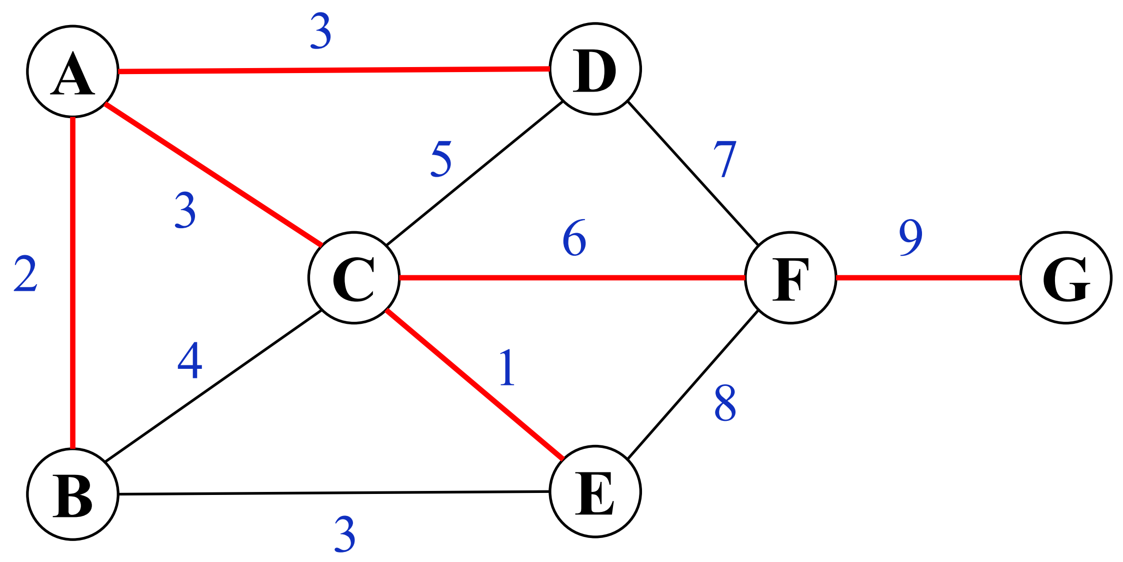


$\{A, B\}, \{A, C\}, \{C, E\}, \{A, D\}, \{C, F\}, \{F, G\}$

2 3 1 3 6 9

MST of weight: 24

Kruskal's Algorithm



$\{C,E\},\{A,B\},\{A,C\},\{A,D\},\{C,F\},\{F,G\}$

1 2 3 3 6 9

MST of weight: 24

Comparing the Prim's and Kruskal's Algorithms

Both algorithms choose and add at each step a min-weight edge from the remaining edges, subject to constraints

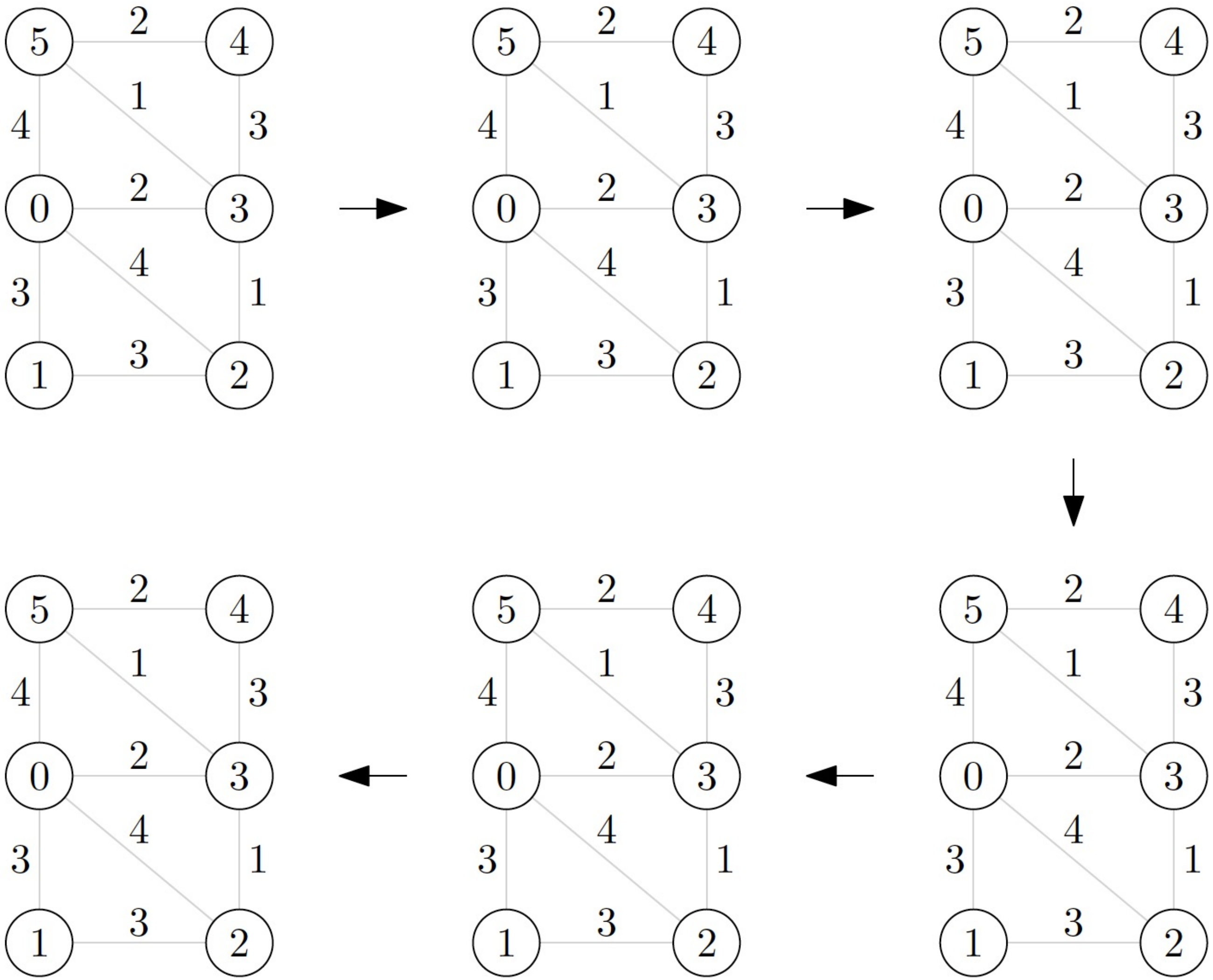
Prim's MST algorithm:

- Start at a root vertex.
- Two rules for a new edge:
  1. No cycle in the subgraph built so far.
  2. Connect the subgraph built so far.
- Terminate if no more edges to add can be found.
- At each step: an acyclic connected subgraph being a tree.

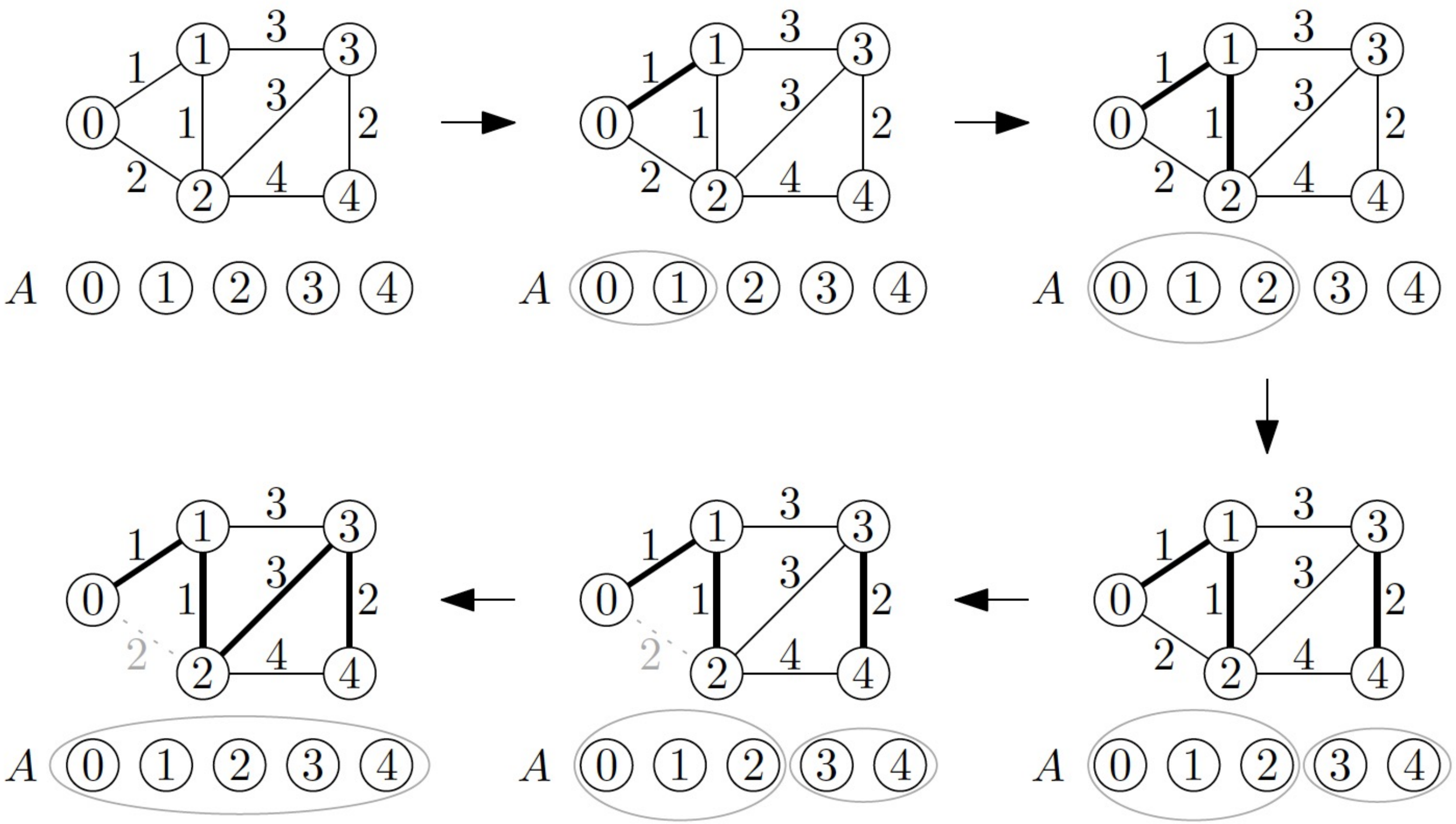
Kruskal's MST algorithm:

- Start at a min-weight edge.
- One rule for a new edge:
  - No cycle in a forest of trees built so far.
- Terminate if no more edges to add can be found.
- At each step: a forest of trees merging as the algorithm progresses (can find a spanning forest for a disconnected graph).

Example 32.4. Execute Prim's algorithm



Example 32.6. Application of Kruskal's algorithm on a graph shown until an MST is found. Note that the edge {0, 2} with weight 2 is not added, because 0 and 2 are already in the same set in A.



SUMMARY

- Terminology
  - Spanning Trees
  - Minimum Spanning Trees (MST)

- Finding MST es Algorithms
  - Prim’s Algorithm
  - Kruskal's Algorithm
  - Time Complexity Analysis
  - Comparison