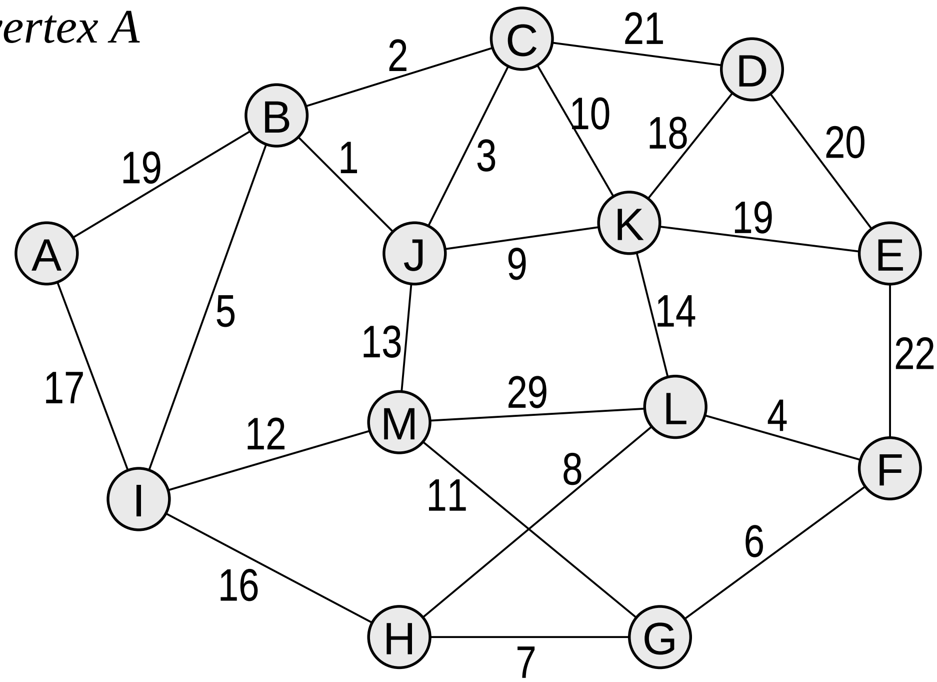
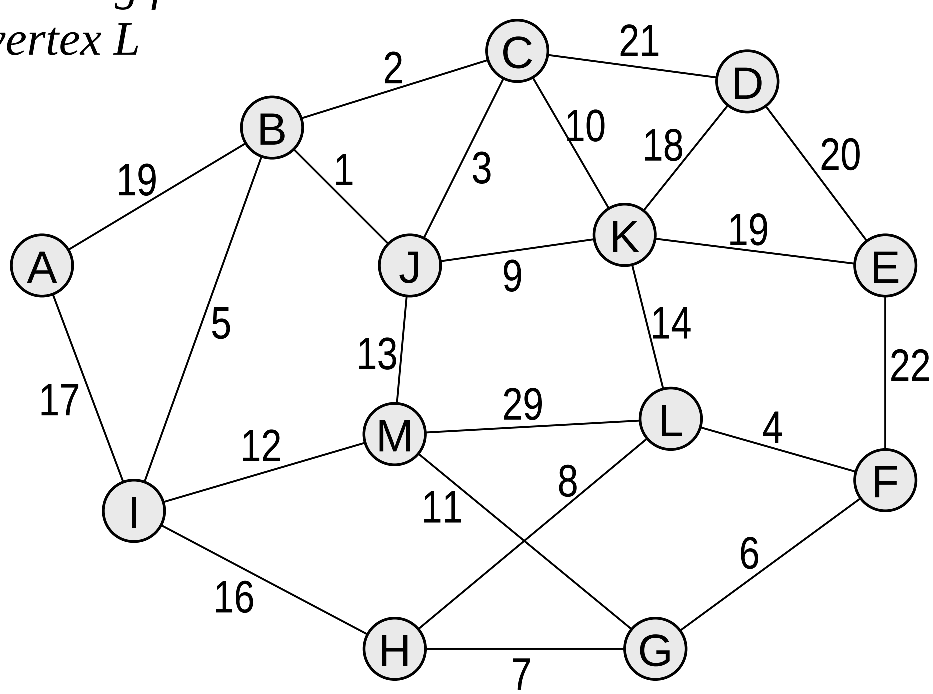
Prim's Minimal Spanning Tree Algorithm

Starting from vertex A



vert	edge	wt
A	---	---

Starting from vertex L



vert	edge	wt
L	---	---

Prim's Algorithm (Minimal Spanning Tree)

Input: A (undirected) weighted graph $G = (V, E, W)$, that is connected. We let $n = |V|$ and $e = |E|$.

Output: A subset E' of E such that $T = (V, E', W)$ is a minimal spanning tree for G.

Algorithm: Start with a single vertex. Repeatedly choose the cheapest edge leading from a vertex already chosen to one not yet chosen. Choose the new vertex to which this edge leads.

Here is a crude implementation using Θ(n³) time.

```
1. SetOfEdges prim( WeightedGraph G)
2.     Choose any vertex v;
3.     V′ = {v};  E′ = ϕ;
4.     while ( V′ ⊂ V )
5.         Among all pairs (x,y) with x ∈ V–V′ and y ∈ V′,
                   choose (x,y) to minimize W(xy);
6.         V′ = V′ ∪ {x};  E′ = E′ ∪{xy};
7.     return E′;
```

On the k^{th} pass through the loop, $|V'| = k$ in line 5, so we are minimizing over $k(n-k)$ pairs. This requires time about $ck(n-k)$, c constant. Summing over $k = 1, 2, \ldots, n$, we obtain $\Theta(n^3)$ total time for line 5, and for the algorithm.

Here is a faster implementation using Θ(n²) time.

An array *near*[] is used to avoid performing the same computations repeatedly in line 5 of the crude version. For each vertex w of $V-V'$, *near*[w] will hold the vertex in V' closest to w.

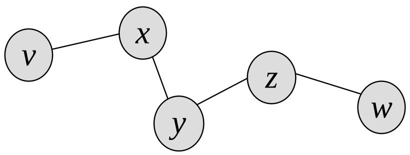
```
1.  SetOfEdges prim( WeightedGraph G)
2.      Choose any vertex v;
3.      V′ = {v};  E′ = ϕ;
4.      for ( each vertex w in V–{v} )
5.          dist[w] = ∞;
6.      for ( each vertex x adjacent to v )
7.          near[x] = v;  dist[x] = W(vx);
8.      while ( V′ ⊂ V )
9.          Choose a vertex x in V–V′ to minimize dist[x];
10.         V′ = V′ ∪ {x};  E′ = E′ ∪ { near[x] x };
11.         for ( each vertex y of V–V′ adjacent to x )
12.             if ( W(xy) < W( near[y] y) )
13.                 near[y] = x;  dist[y] = W(xy);
14.     return E′;
```

Lines 4-5 require $\Theta(n)$ time. Lines 6-7 combined with all passes over lines 11-13 traverse each adjacency list once, performing a constant amount of work for each entry, so the total time for these lines is $\Theta(e)$ with an adjacency list ($\Theta(n^2)$ with an adjacency matrix). Line 9 uses $\Theta(n)$ time on each pass, or a total of $\Theta(n^2)$. The total running time is $\Theta(n^2)$.

Dijkstra's Single Source Shortest Path Algorithm

The problem: Given a weighted graph or digraph $G = (V,E,W)$, and a fixed vertex v, find the distances and shortest paths from v to every other vertex. (We assume all weights are positive; $short(v,w)$ denotes shortest path from v to w.)

Idea:



If v,x,y,z,w is the shortest path from v to w, then

i) v,x,y,z is the shortest path from v to z,

ii) $dist(v,z) < dist(v,w)$,

iii) $dist(v,w) = dist(v,z) + W(zw)$.

$$\left.\begin{array}{l} short(v,w) = short(v,z),\, w \\ dist(v,w) = dist(v,z) + W(zw) \end{array}\right\}$$ for some vertex z, adjacent to w, with $dist(v,z) < dist(v,w)$.

Which vertex z? Among all possible z, that which minimizes $dist(v,z) + W(zw)$.

If we already know the k closest vertices to v, and their distances from v, the k+1st closest vertex may be found like this:

T = { k closest vertices to v, including v itself (*tree* vertices)},

F = { vertices of $V-T$ adjacent to vertex in T (*fringe* vertices)}.

Choose $z \in T$ and $w \in F$ to so

$$dist(v,z) + W(zw) = \min\{\, dist(v,t) + W(tf) : t \in T, f \in F \,\}.$$

Then

w is the k+1st closest vertex to v,

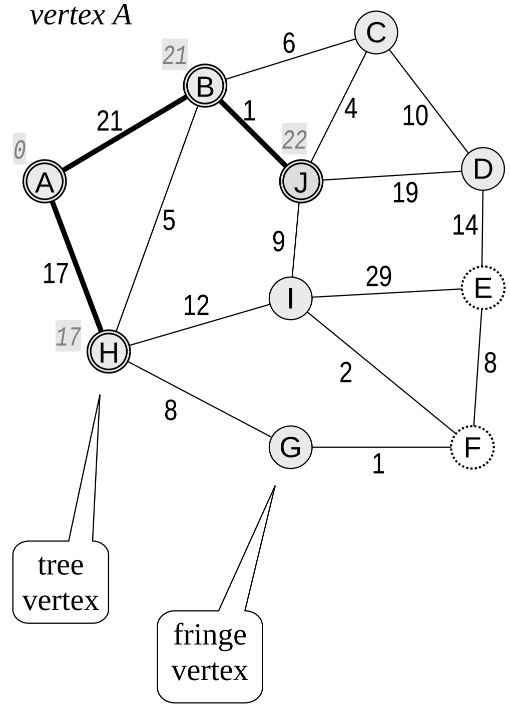
$dist(v,w) = dist(v,z) + W(zw)$,

$short(v,w) = short(v,z),w$

A straightforward implementation would take $\Theta(n^2)$ time to find a single pair (z,w) above, and hence $\Theta(n^3)$ time to find the distance from v to all other vertices.

But a technique very similar to that used to speed up Prim's algorithm works here — and reduces the total time to $\Theta(n^2)$.

Distances from vertex A



t = tree vertex	f = fringe vertex	$dist(v,t) + W(tf)$
B	C	27
J	C	26
J	D	41
J	I	31
H	***G***	**25**
H	I	29

Minimum occurs for (H,G). Fifth closest vertex is G, and $dist(A,G) = 25$.

Note: $dist(\mathrm{A,D}) \neq 41$, $dist(\mathrm{A,I}) \neq 29$.