Connectivity

A *path* of *length* n from u to v is a sequence of vertices $x_0, x_1, \ldots, x_n$, where x_{i-1} to x_i is an edge for each i, $1 \leq i \leq n$.

A *circuit* or *cycle* is a path that begins and ends at the same vertex.

A path is *simple* if it does not contain the same vertex more than once.

A simple graph is *connected* if there is a path between every pair of vertices.

A directed graph is *strongly connected* if there is a path between every pair of vertices.

A directed graph is *weakly connected* if there is a path between every pair of vertices in the underlying simple graph.

Let G be a graph with adjacency matric $\mathbf{A}$. The number of paths of length n from v_i to v_j is the (i, j) entry of $\mathbf{A}^n$.

Here is a proof by mathematical induction.

Predicate $P(n)$: The number of paths of length n from v_i to v_j is the (i, j) entry of $\mathbf{A}^n$.

Basis $P(1)$: True for $\mathbf{A}^1$ because $a_{ij} = 1$ iff there is an edge from v_i to v_j.

Induction: Prove $P(k) \rightarrow P(k+1)$

Assume $P(k)$. Show $P(k+1)$.

Proof: Let $\mathbf{B} = \mathbf{A}^k$. $\mathbf{A}^{k+1} = \mathbf{B}\,\mathbf{A}$, so the (i, j) entry of $\mathbf{A}^{k+1}$ is $b_{i1}\,a_{1j} + b_{i2}\,a_{2j} + \cdots$

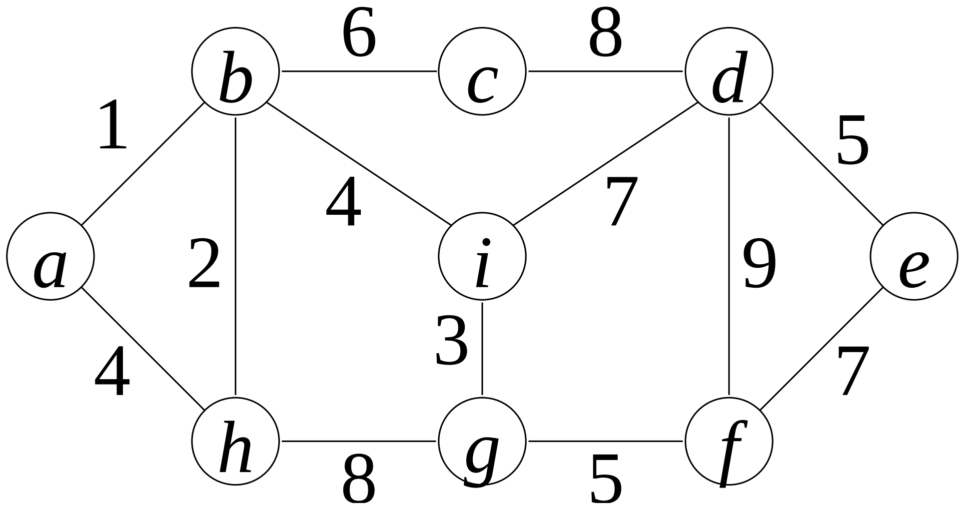
This adds the paths of length k from v_i which can be followed by an edge to v_j.

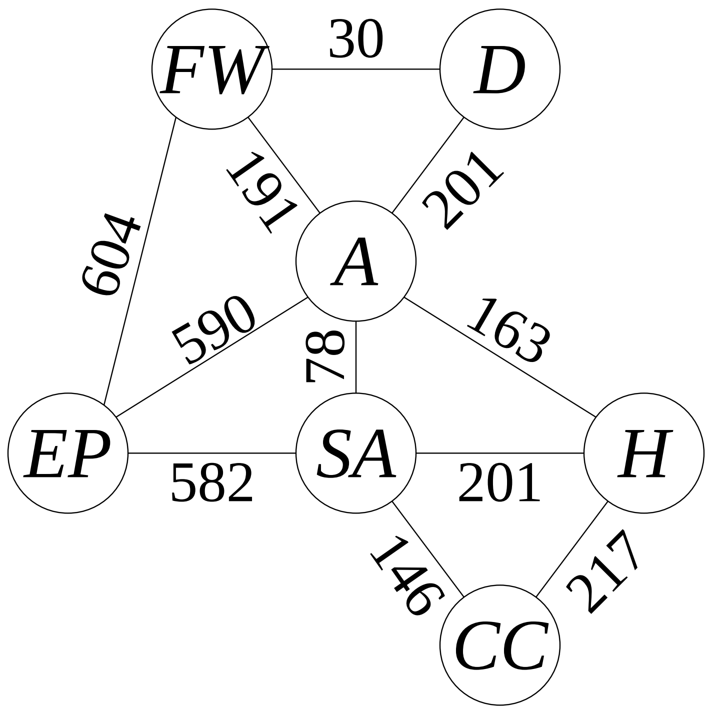
Weighted Graphs

A weighted graph is a graph in which each edge (u, v) has a weight $w(u, v)$. Each weight is a real number. Weights can represent distance, cost, time, capacity, etc.

The *length* of a path in a weighted graph is the sum of the weights on the edges.

Dijkstra's Algorithm finds the shortest path between two vertices.


a
b
c
d
e
f
g
h
i
1
6
8
5
4
7
2
9
3
4
7
8
5


FW
D
A
EP
SA
H
CC
30
191
201
604
590
78
163
582
201
146
217

```
procedure shortest_path
(G: weighted simple graph; a, z: vertices in G)
    {Assume that a is connected to z}
    {L(v) is min. known distance from a to v}
    {S is the set of processed vertices}
    for each vertex v in G
        L(v) := ∞
    L(a) := 0
    S := ∅
    loop
        u := vertex not in S with min. L value
        if u = z then exit loop
        S := S ∪ {u}
        N := (neighbors of u) − S
        for each vertex v in N
            if L(u) + w(u, v) < L(v)
            then L(v) := L(u) + w(u, v)
    return L(z)
```