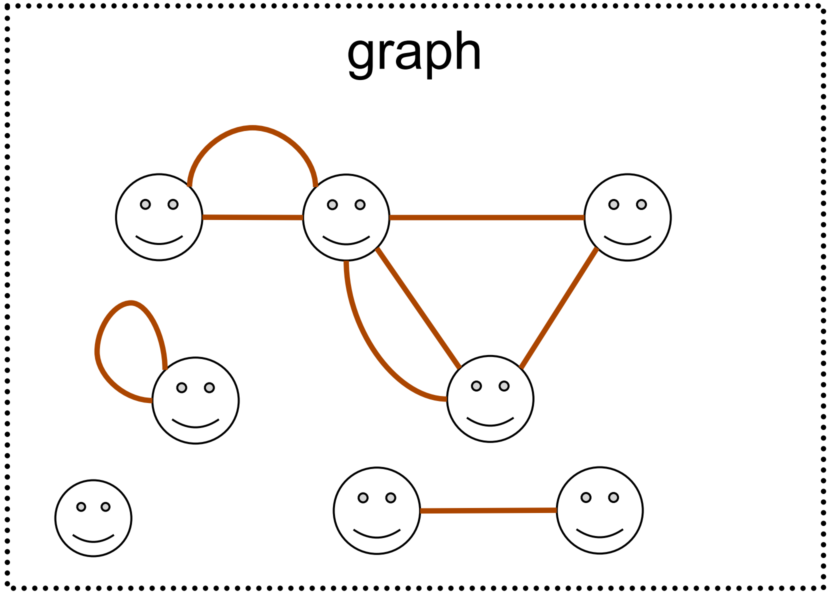
Concepts of graph theory

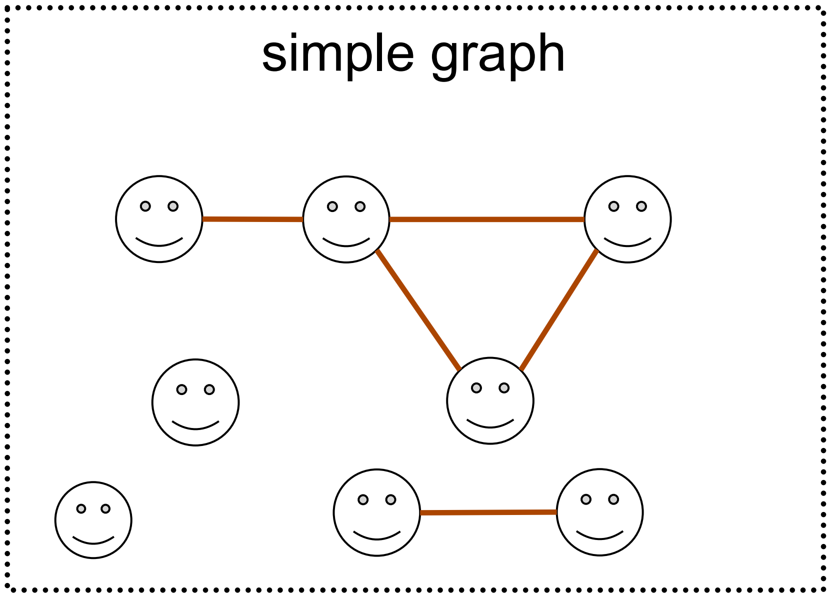
Contents

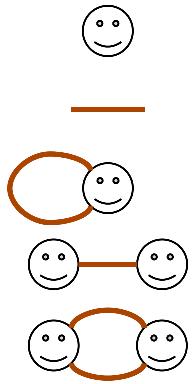
Basic concepts of graph theory

Graph definitions

Graph







vertex (***node***)

edge

self-loop

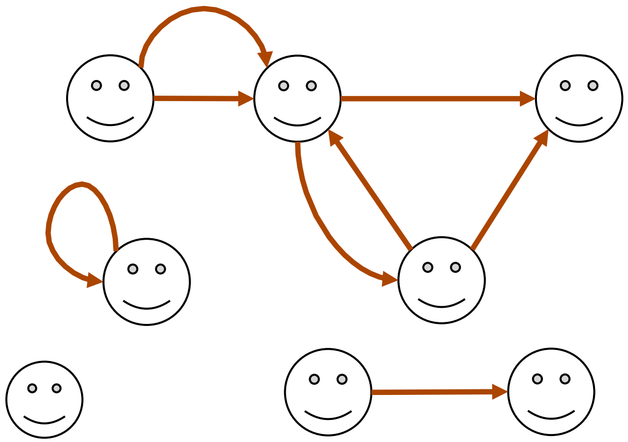
proper edge

multi-edge

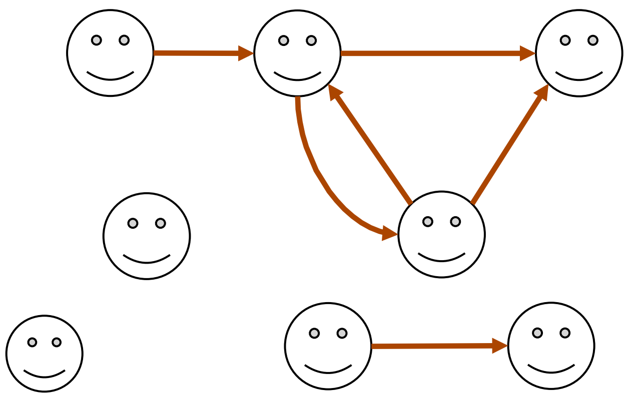
- A ***grapH (***G) contains a set of ***vertices*** (V) and a set of ***edges*** (E)
- A ***simple graph*** contains no self-loop and no multi-edge

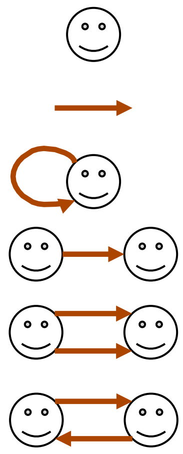
Directed GrapH (= Digraph)

digraph



simple digraph





vertex (= ***node***)

arC (= directed edge)

self-loop

proper arc

multi-arc

not a multi-arc !

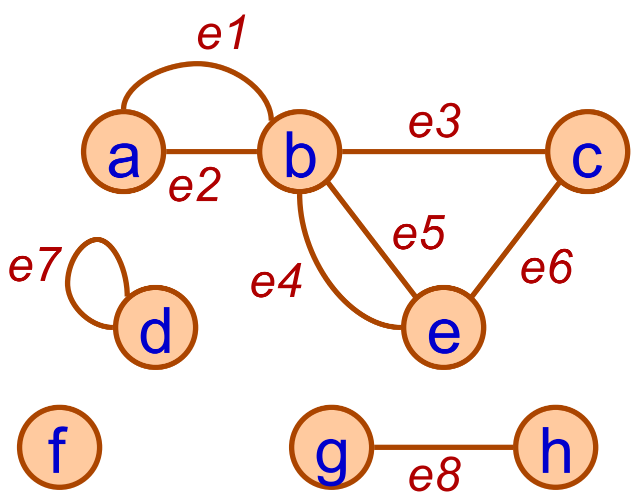
- A ***directed edgD (***or ***arc***) is characterized by a ***head*** and a ***tail***
- A ***digraph*** is a graph whose edges are directed
- A ***partially directed graph*** is a graph combining directed and non-directed edges

Graph descriptions

Graph descriptions : incidence matrix

- one row per edge
- one colum per vertex
- value = 1 if edge and vertex are ***incident***
- **Problems**
  - only valid for undirected graphs
  - inefficient storagD (many empty cells)

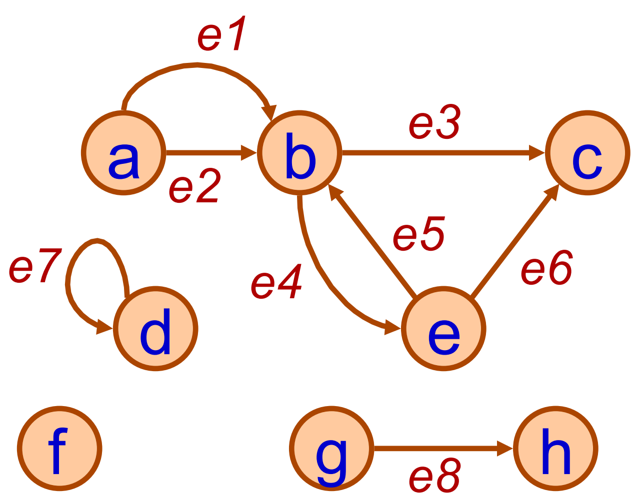
edge\vertex	a	b	c	d	e	f	g	h
e1	1	1	0	0	0	0	0	0
e2	1	1	0	0	0	0	0	0
e3	0	1	1	0	0	0	0	0
e4	0	1	0	0	1	0	0	0
e5	0	1	0	0	1	0	0	0
e6	0	0	1	0	1	0	0	0
e7	0	0	0	0	1	0	0	0
e8	0	0	0	0	0	0	1	1



Graph descriptions : adjacency matrix

- one row per vertex
- one colum per vertex
- value = 1 if vertices are ***adjacent***
- diagonal = self-loops
- **Problems**
  - no possibility to represent multi-arcs
  - inefficient storagD (many empty cells)

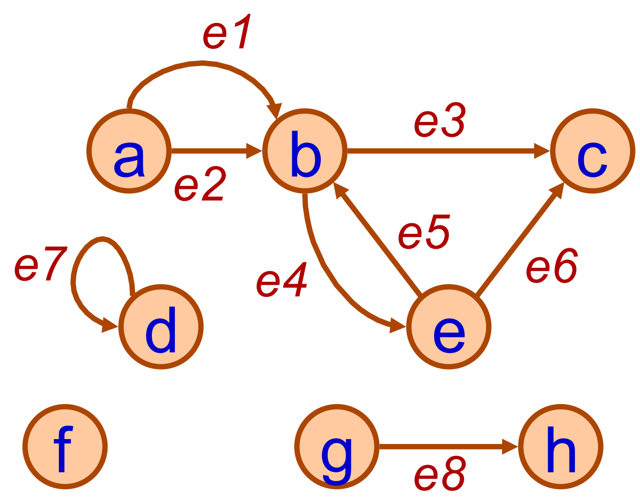
from\to	a	b	c	d	e	f	g	h
a	0	1	0	0	0	0	0	0
b	0	0	1	0	1	0	0	0
c	0	0	0	0	0	0	0	0
d	0	0	0	1	0	0	0	0
e	0	1	1	0	0	0	0	0
f	0	0	0	0	0	0	0	0
g	0	0	0	0	0	0	0	1
h	0	0	0	0	0	0	0	0



Graph descriptions : adjacency list

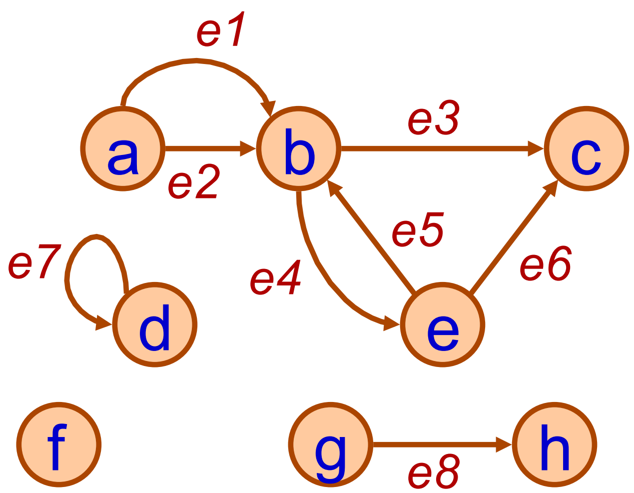
- A list of out-going vertices is associated to each vertex
- Compact representation
- Optionally, a list of in-going vertices can be added to allow reverse-traversal of the graph

Vertex	*out*	*in*
a	(b,b)	()
b	(c,e)	(a,e)
c	()	(b,e)
d	(d)	(d)
e	(b,c)	(b)
f	()	()
g	(h)	()
h	()	(h)



Graph descriptions : formal description

- one row per edge
- one column for heads
- one column for tails
- optional columns for edge attributes (label, weight, color, …)

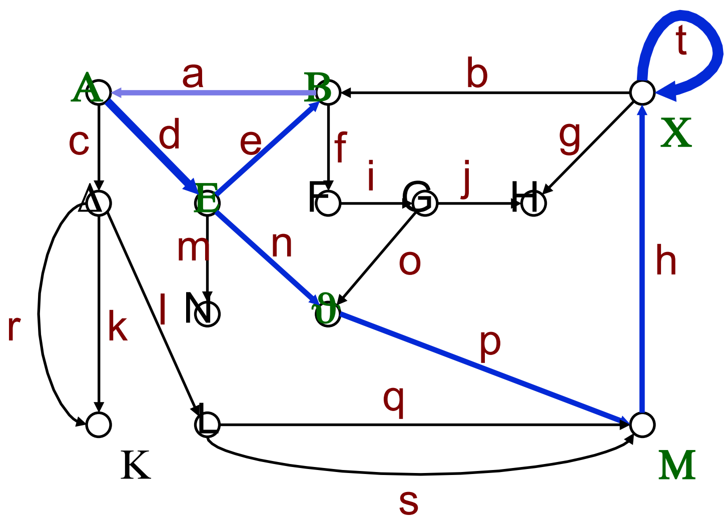


head	tail	label
a	b	e1
a	b	e2
b	c	e3
b	e	e4
e	b	e5
e	c	e6
d	d	e7
g	h	e8

Walks, trails and paths

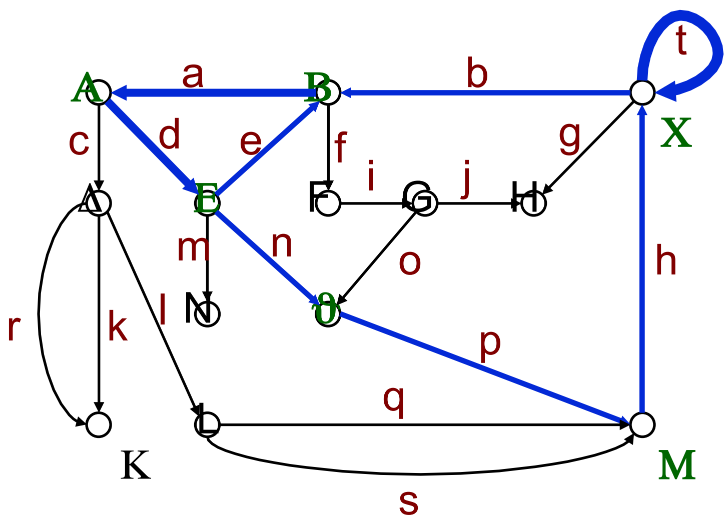
Walk

- A ***walk*** from vertex A to vertex B is an alternating sequence of vertices and edges, representing a continuous traversal from A to B
- Remarks
  - A walk can be described unequivocally by the sequence of edges (e.g.: **d, e, a, d, n,p,h,t,t,t**)
  - In a non-simple grapH (i.e. with multi-edges), a walk is not described unequivocally by a sequence of vertices
  - An edge or a vertex can appear repeatedly in the same walk (e.g.: edges **d** and **t** , and vertices α, ε, χ on the figure)



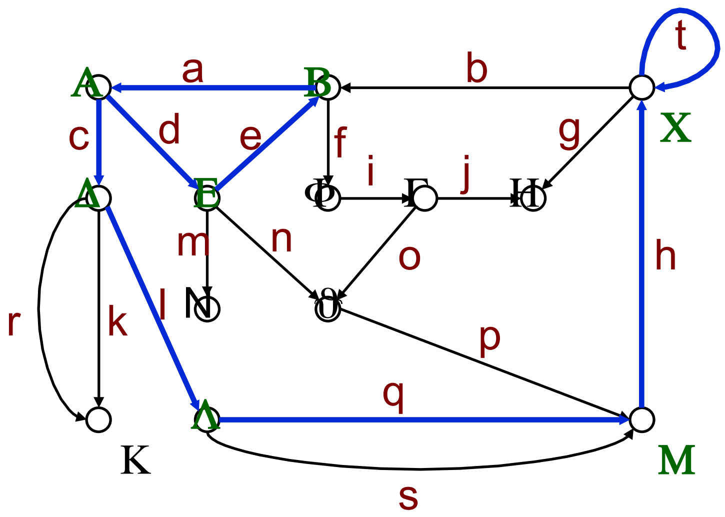
Closed walk

- A ***closed walk*** is a walk whose initial and final vertices are identical (e.g.: **d, e, a, d, n,p,h,t,t,t,b,a**)



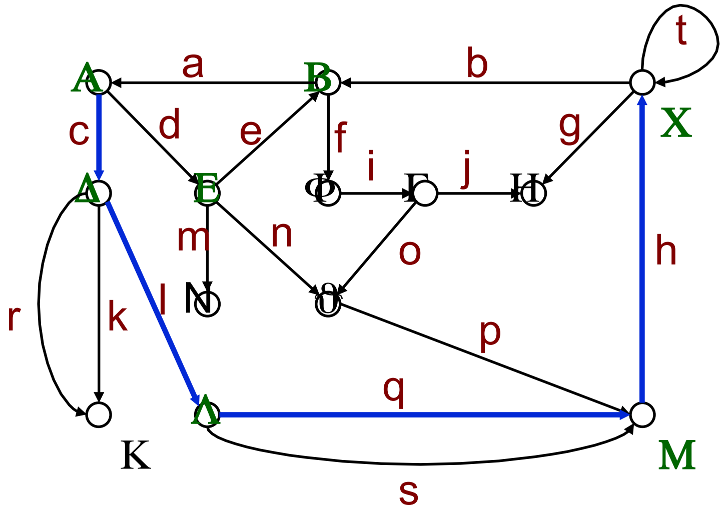
Trail

- A ***trail*** is a walk with no repeated edges (e.g.: **d, e, a, c,l,q,h,t**)
- Remark: a vertex can appear repeatedly in the same traiL (e.g.: α and χ on the figure)

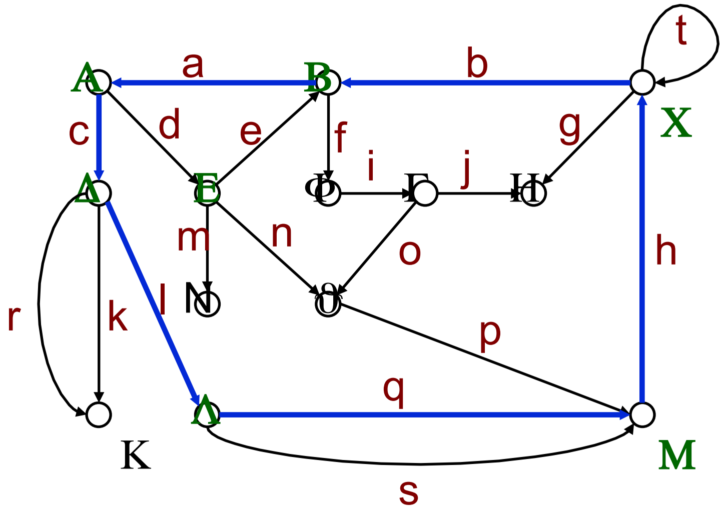


Path

- A ***path*** is a trail with no repeated vertices, except possibly the initial and final vertex (e.g. **c,l,q,h**)



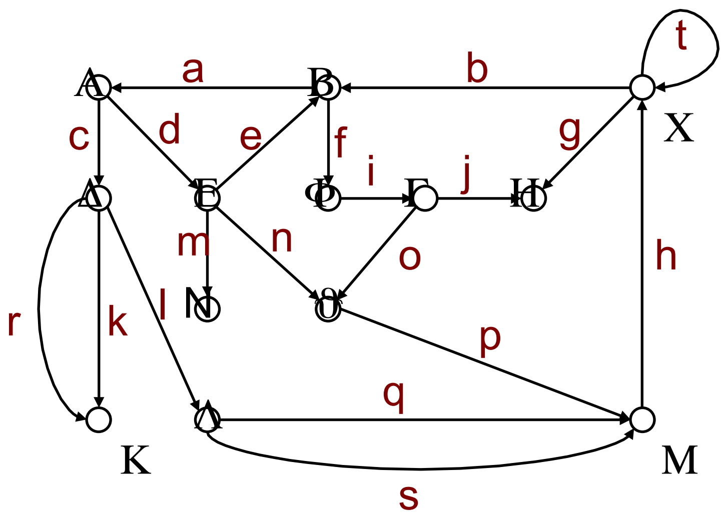
- A ***cycle*** is a closed path with at least one edge (e.g. **c,l,q,h,b,a**)



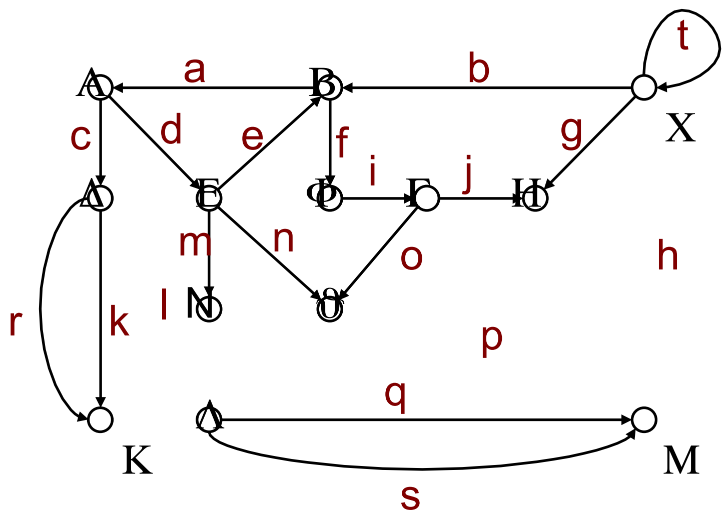
Connected graph

- a ***connected graph*** is a graph in which there is a walk between every pair of distinct vertices

Connected graph



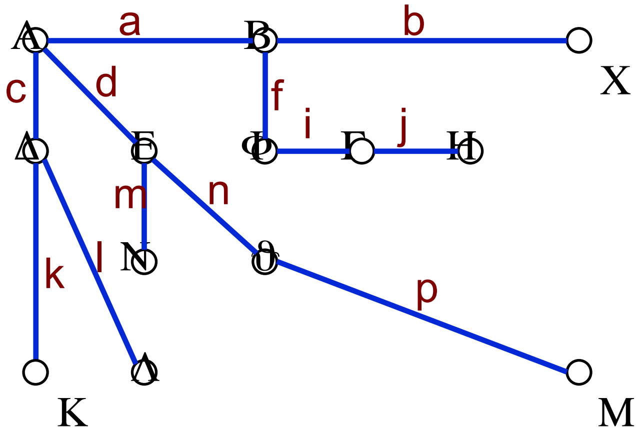
Non-connected graph



Trees

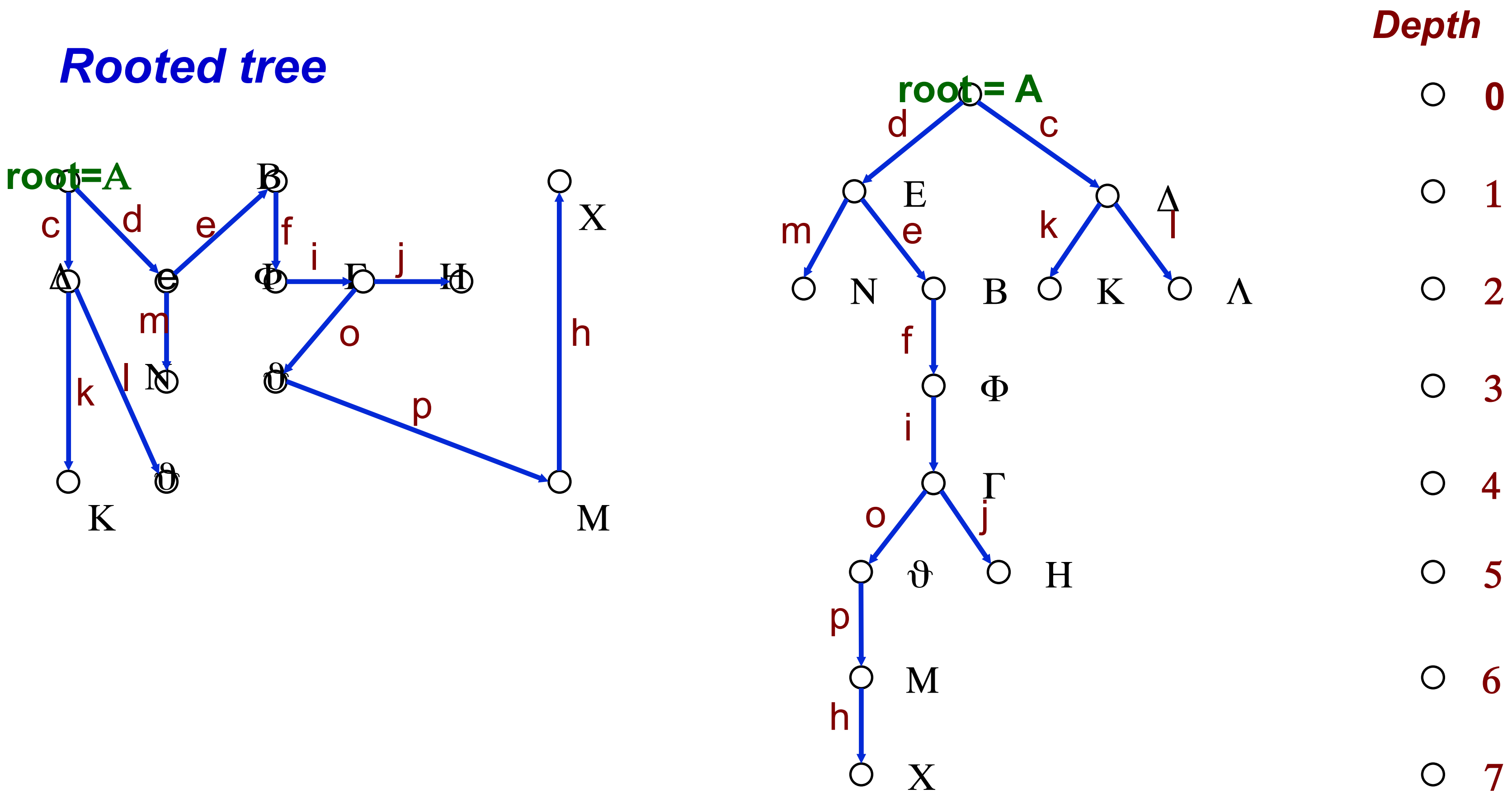
- a ***tree*** is a connected graph that has no cycles

Tree

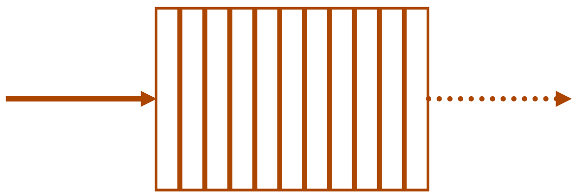


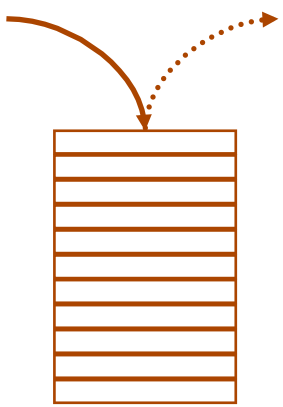
Rooted tree

- A ***rooted tree*** is a directed tree having a distinguished vertex *r* called the ***root*** such that for each other vertex *v*, there is a directed path from the root to *v*
- Each non-root node has a single parent



Queue and stack



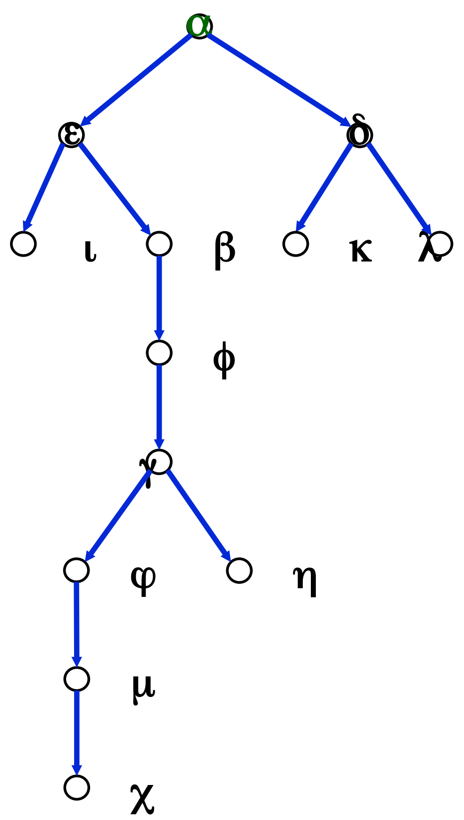


- A ***queue*** is a sequence of elements such that each new element is addeD (***enqueued***) to one end, called the ***back*** of the queue, and an element is removeD (***dequeued***) from the other end, called the ***front***

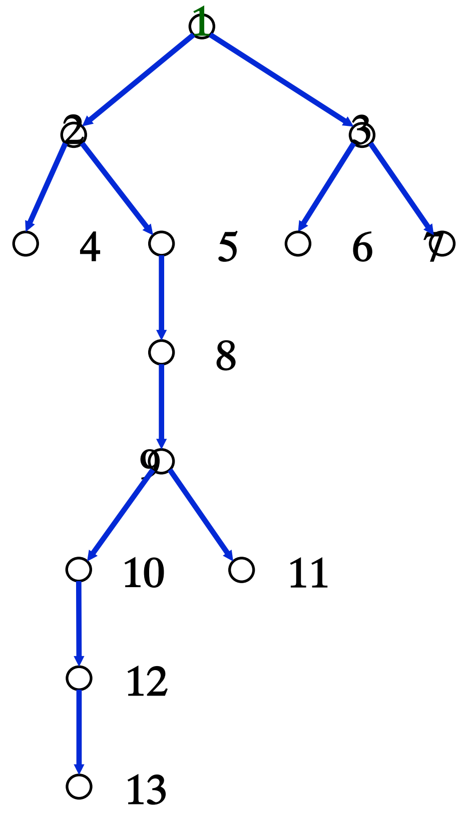
- A ***stack*** is a sequence of elements such that each new element is addeD (or ***pushed***) onto one end, called the ***top***, and an element is removeD (***popped***) from the same end

Level-order tree traversal with a queue

- Enqueue root
- While queue is not empty
  - Dequeue a vertex and write it to the output list
  - Enqueue its children left-to-right

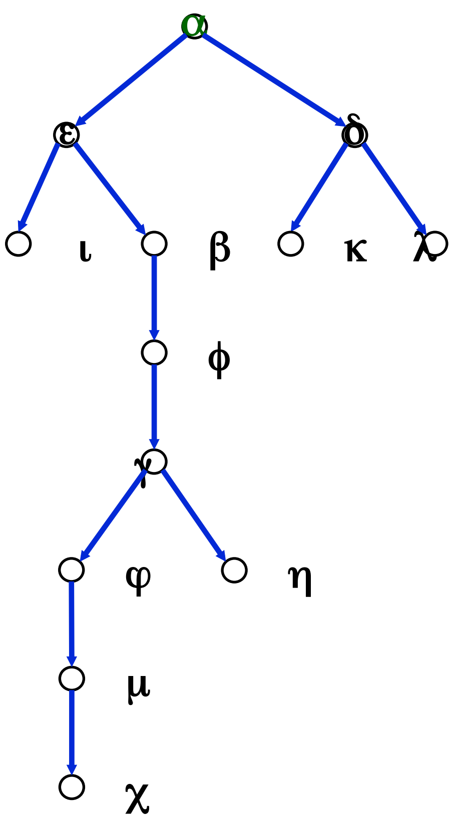


Step	*Output*	*Queue*
0		α
1	α	ε,δ
2	ε	δ,ι,β
3	δ	ι,β,κ,λ
4	ι	β,κ,λ
5	β	κ,λ,ϕ
6	κ	λ,ϕ
7	λ	ϕ
8	ϕ	γ
9	γ	φ,η
10	φ	η,μ
11	η	μ,χ
12	μ	χ
13	χ	

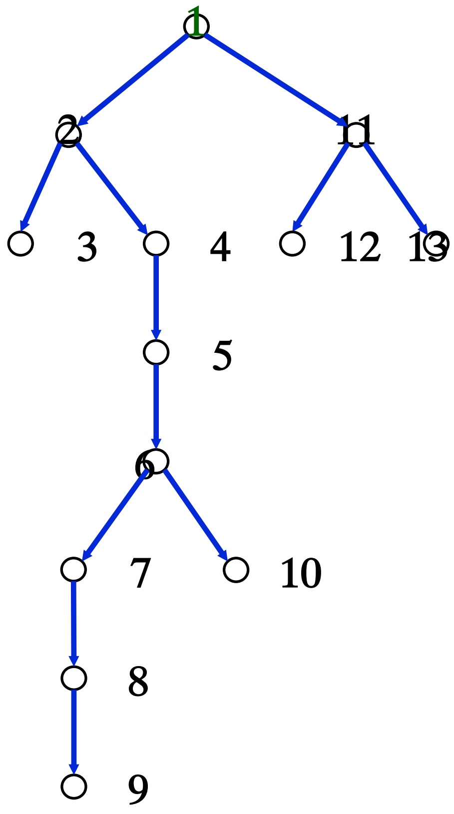


Pre-order tree traversal with a stack

- Push root onto the stack
- While stack is not empty
  - Pop a vertex off stack, and write it to the output list
  - Push its children right-to-left onto stack



Step	*Output*	*Stack*
0		α
1	α	δ,ε
2	ε	δ,β,ι
3	ι	δ,β
4	β	δ,ϕ
5	ϕ	δ,γ
6	γ	δ,η,φ
7	φ	δ,η,μ
8	μ	δ,η,χ
9	χ	δ,η
10	η	δ
11	δ	λ,κ
12	κ	λ
13	λ	



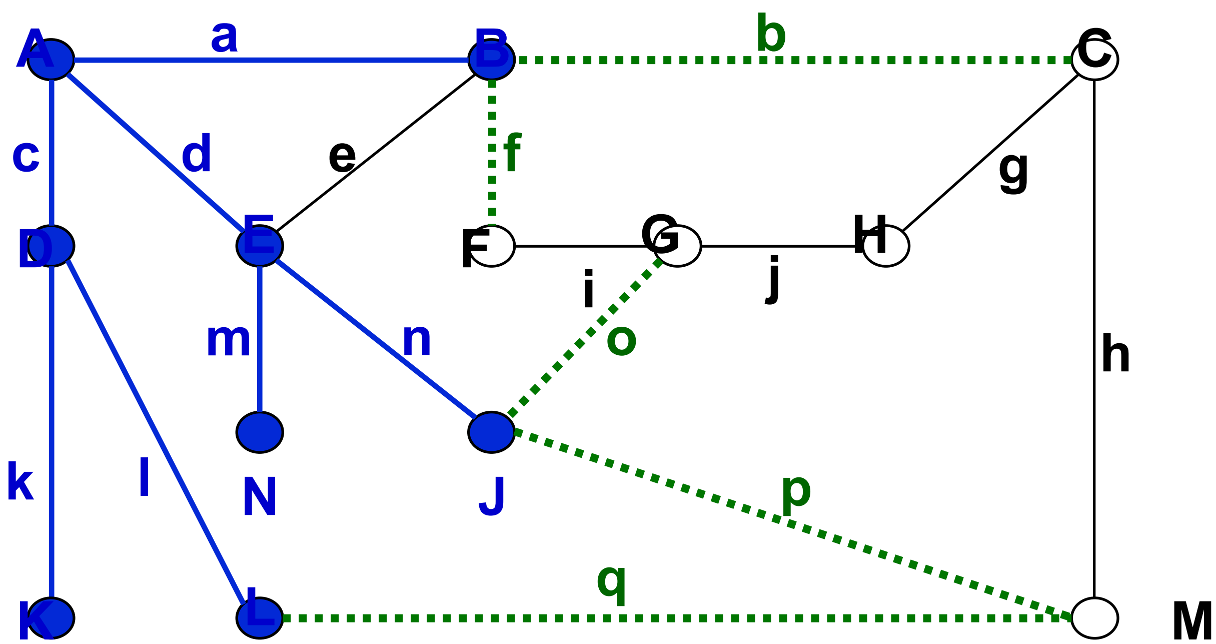
Path finding

Path finding in biochemical networks

- 2-ends path finding
  - Find all pathways from compound A to compound B
- 1-end path finding
  - Find all genes regulated by a membrane receptor via a signal transduction pathway
- 1-end path finding, reverse
  - Find all proteins and compounds exerting a direct or indirect action on the level of expression of a given gene
- Circuit finding
  - Find all feed-back loops
- Subgraph extraction
  - Starting from a set of *n* seed nodes, extract a subgraph that joins “at best” the seeds.
    - Unweighted graphs: minimize the number of edges of the subgrpah
    - Weighted graphs: minimize the weight of the subgraph

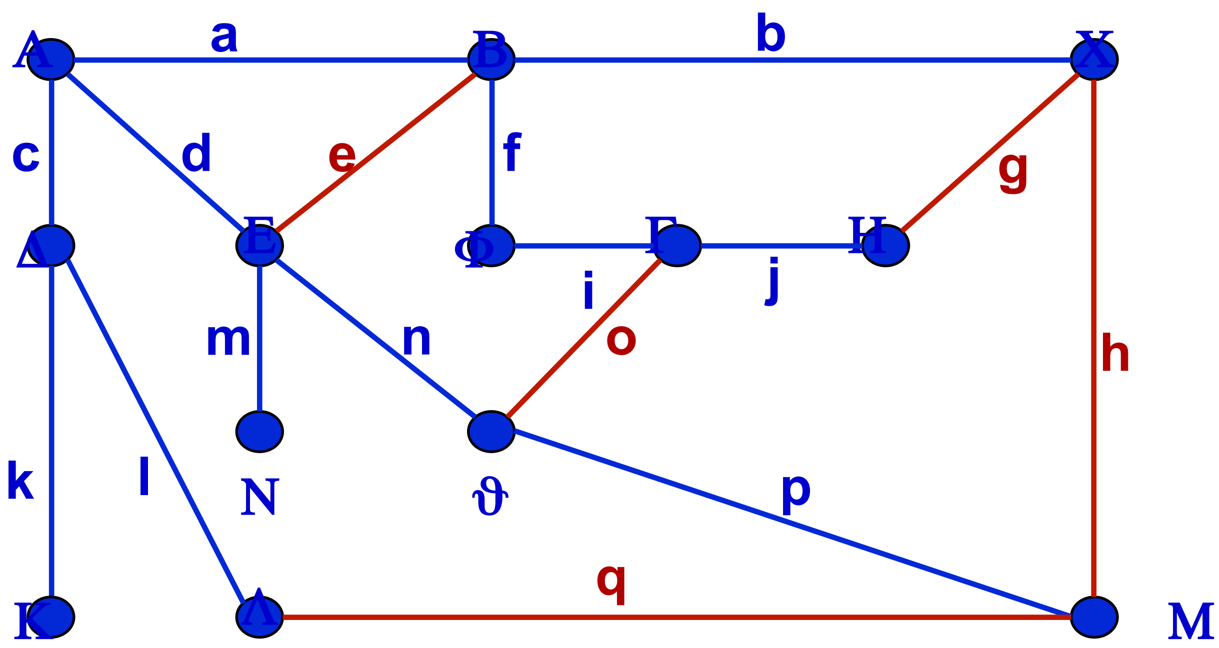
Tree in a graph

- ***A tree T in a graph G*** is a connected subgraph which contains no cycle
- The edges and vertices of T are called ***tree-edges*** and ***tree-vertices***
- A ***frontier-edge*** is a non-tree edge with one endpoint in T and one endpoint not in T.



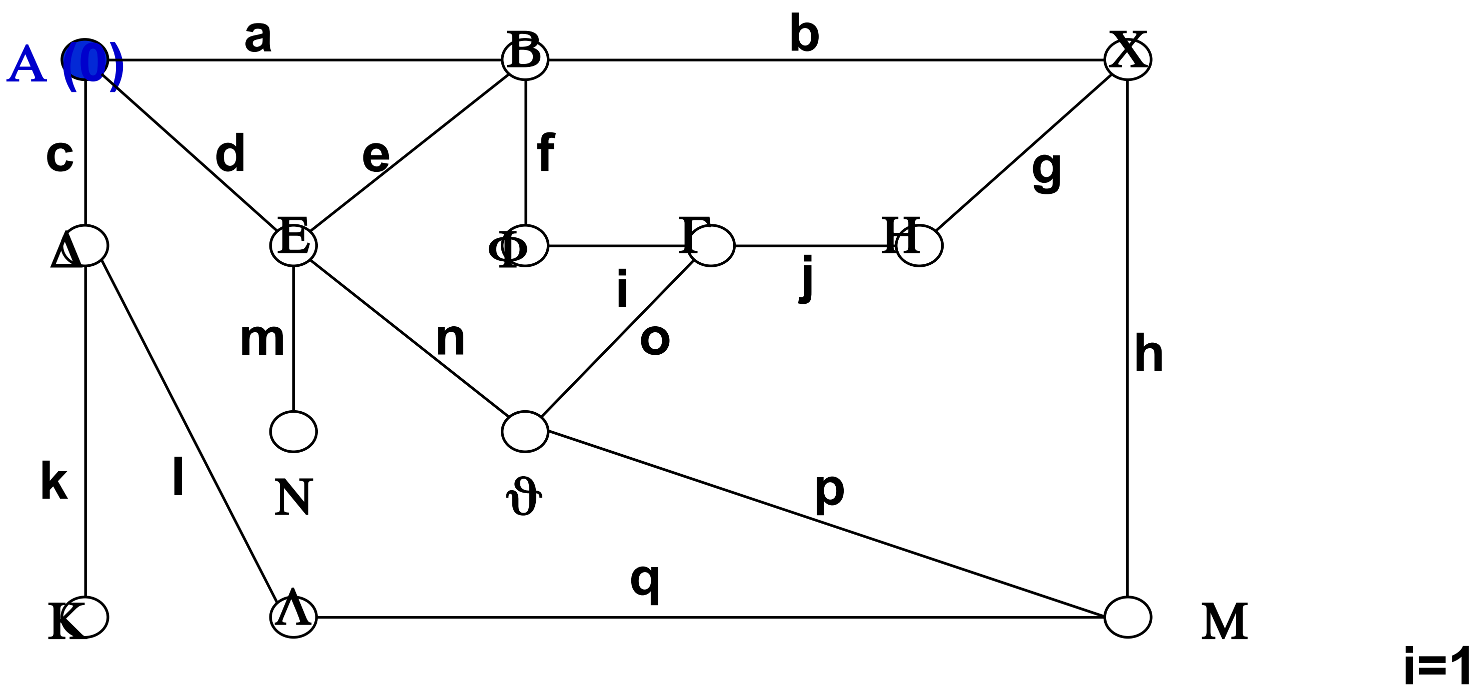
Spanning tree

- A ***spanning tree*** is a tree which contains all the vertices of a graph
- A spanning tree does not necessarily contain all the edges
- A ***fundamental cycle*** in the graph G is the unique cycle which is created when a ***non-tree edge*** is added to the spanning tree T
- Each ***non-tree edge*** corresponds to a fundamental cycle in the graph



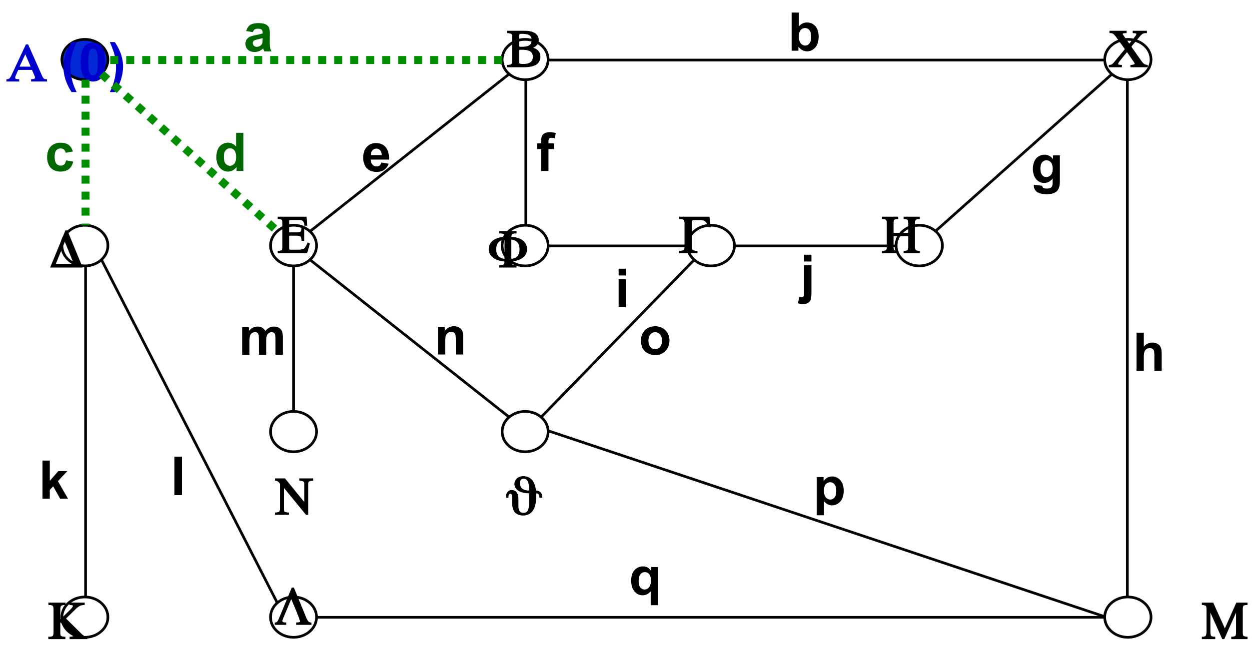
Depth-first-searcH (DFS)

- Initialize tree at a given vertex (for example a)
- Initialize the set of frontier edges as empty
- Set dfnumber(a) to 0
- Initialize label counter i to 1



Depth-first-search

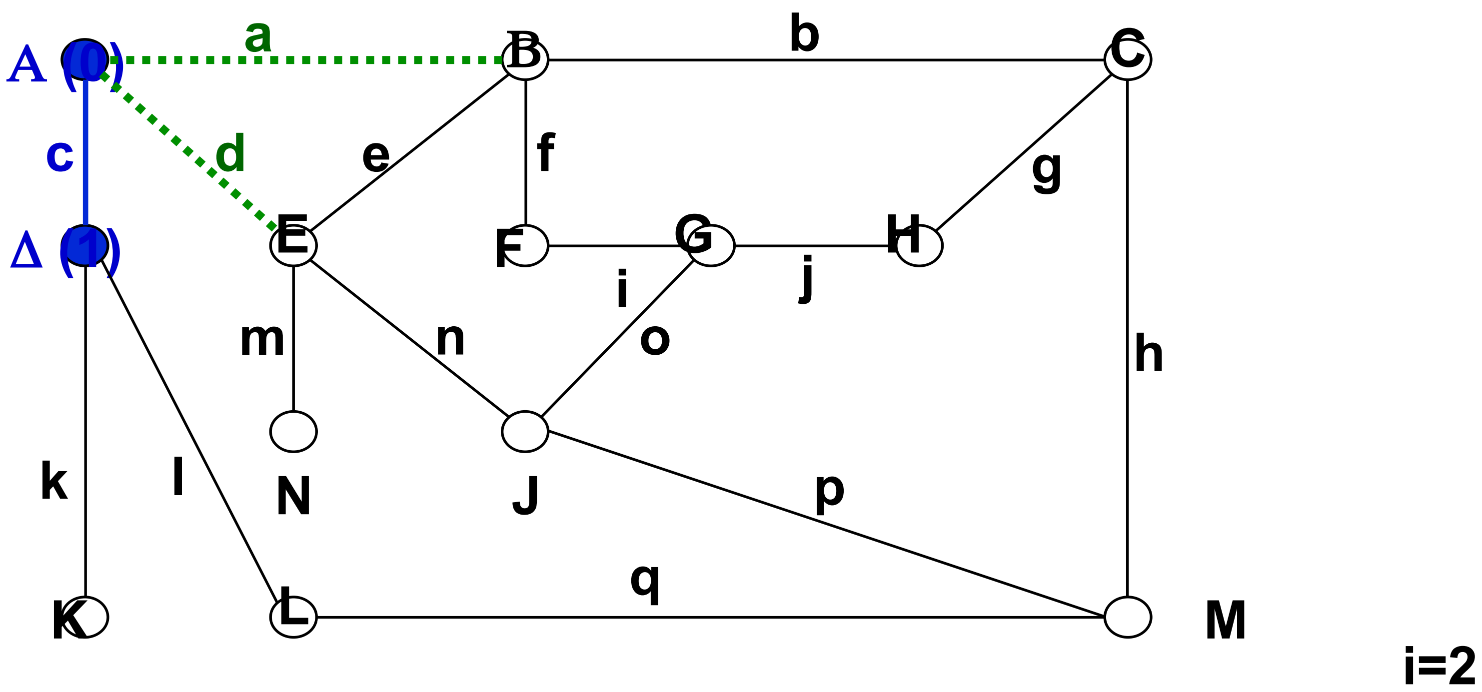
- While T does not span G
  - update the set of frontier edges



i=1

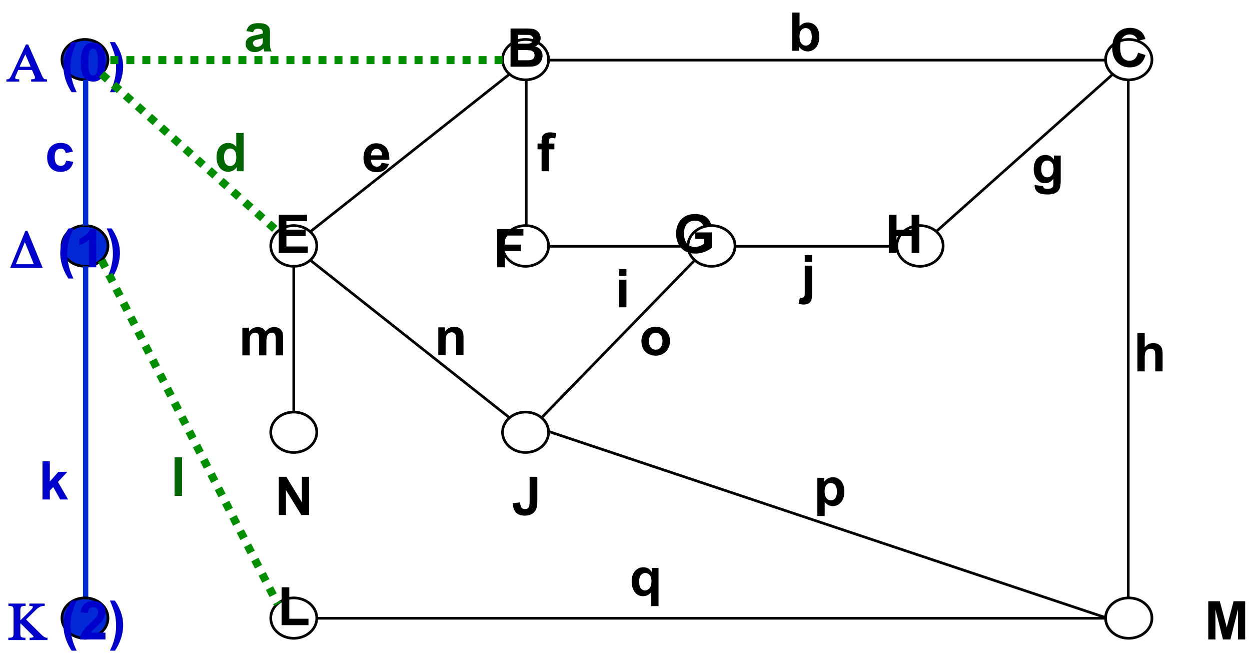
Depth-first-search

- While T does not span G
  - update the set of frontier edges
  - select a frontier edge whose labelled endpoint has the largest possible dfnumber
  - add this edge to the tree
  - select the unlabelled endpoint of this edge, and set its dfnumber to i
  - i := i+1



Depth-first-search

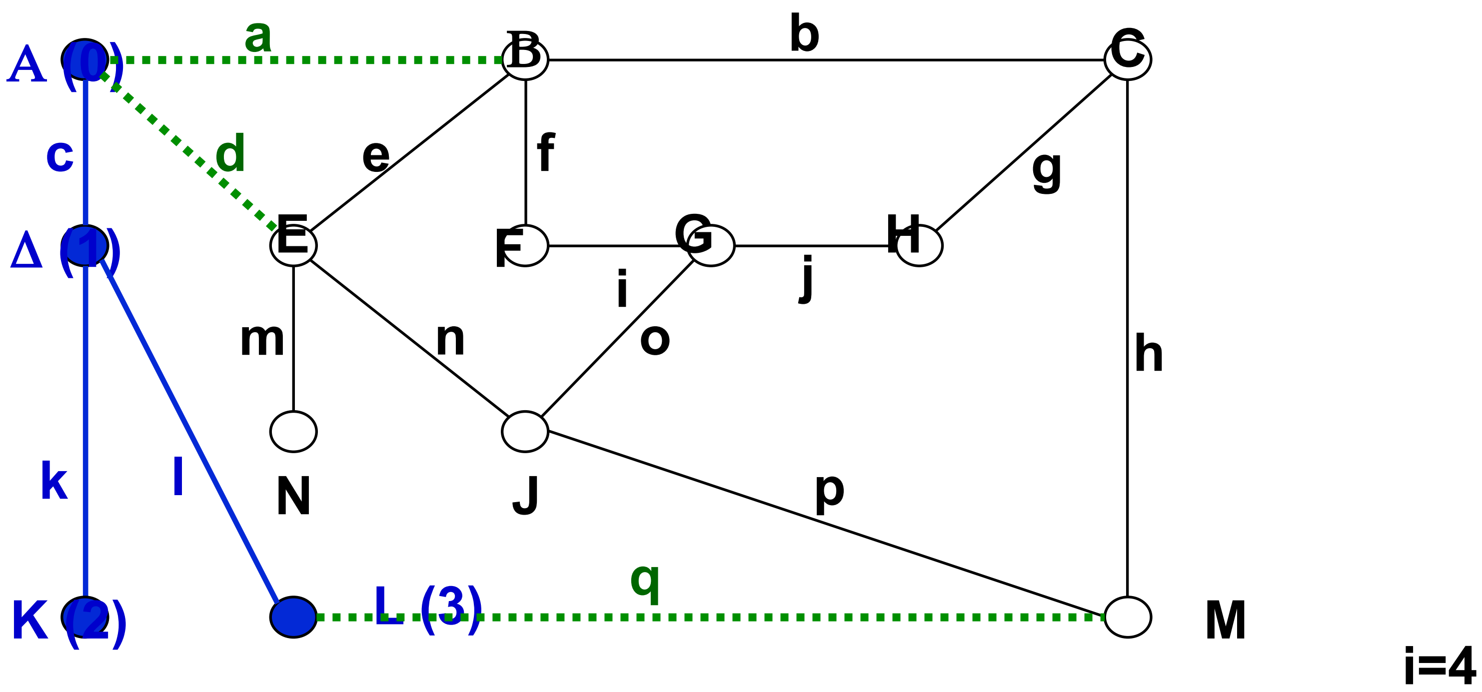
- While T does not span G
  - update the set of frontier edges
  - select a frontier edge whose labelled endpoint has the largest possible dfnumber
  - add this edge to the tree
  - select the unlabelled endpoint of this edge, and set its dfnumber to i
  - i := i+1



i=3

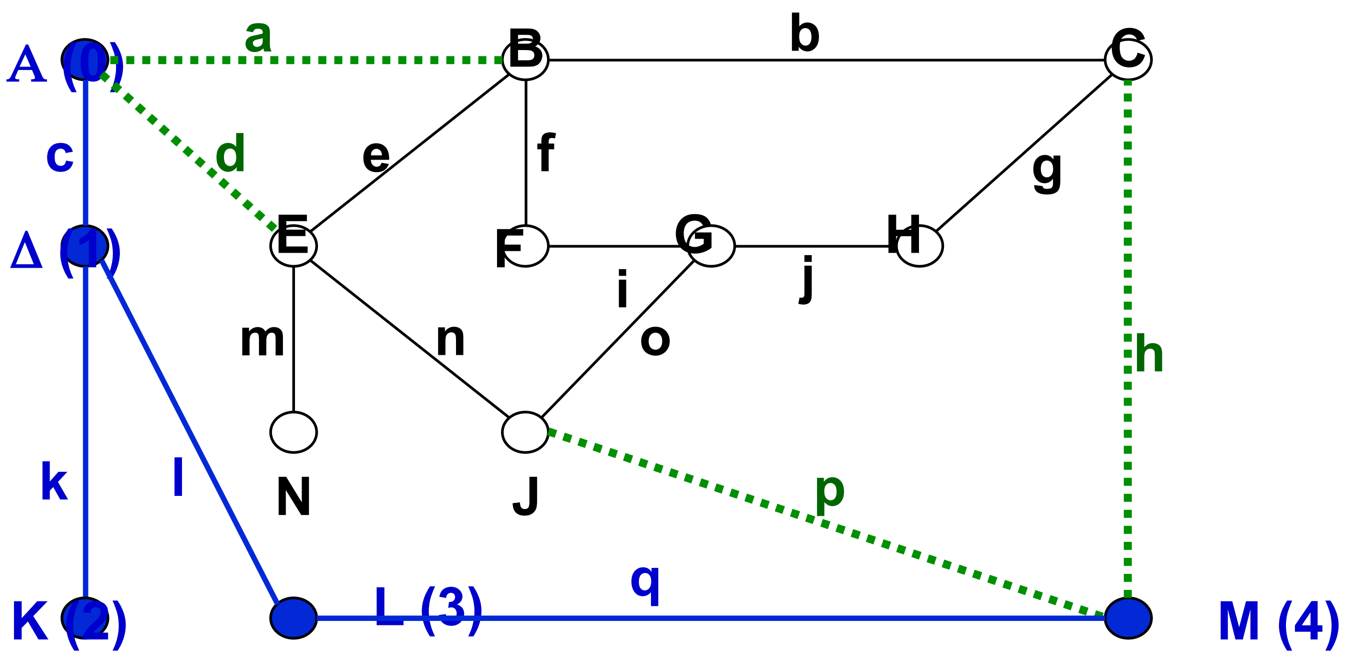
Depth-first-search

- While T does not span G
  - update the set of frontier edges
  - select a frontier edge whose labelled endpoint has the largest possible dfnumber
  - add this edge to the tree
  - select the unlabelled endpoint of this edge, and set its dfnumber to i
  - i := i+1



Depth-first-search

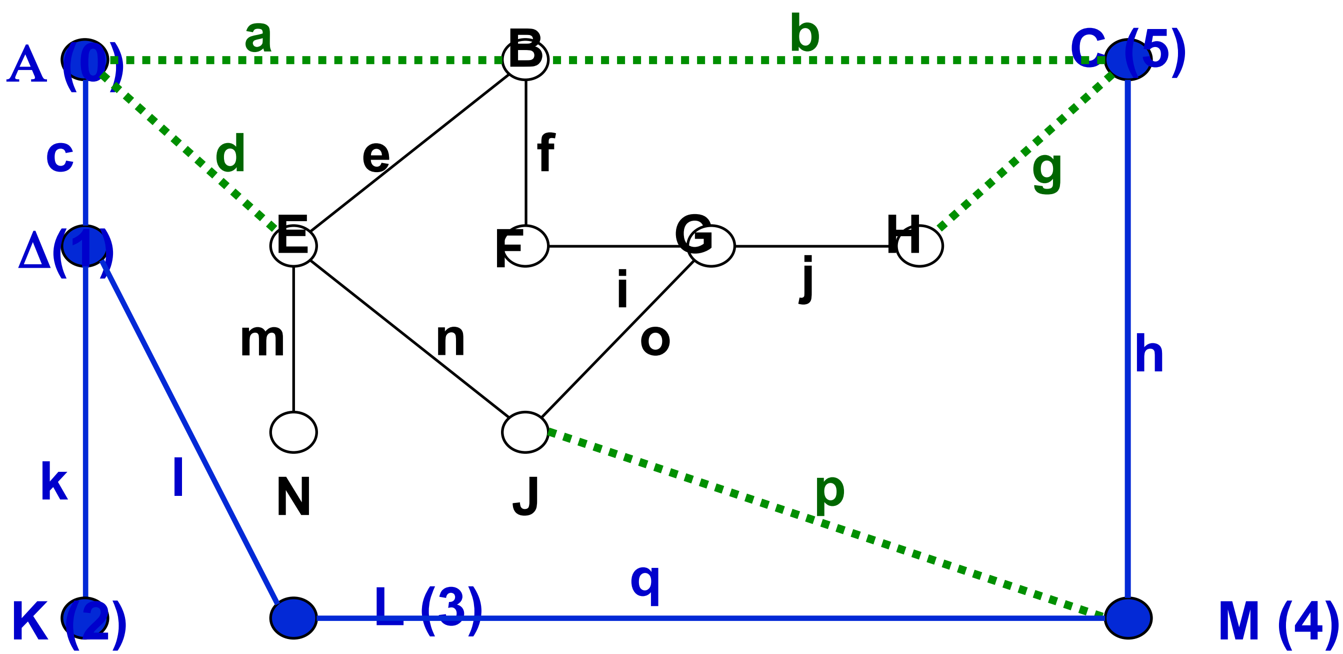
- While T does not span G
  - update the set of frontier edges
  - select a frontier edge whose labelled endpoint has the largest possible dfnumber
  - add this edge to the tree
  - select the unlabelled endpoint of this edge, and set its dfnumber to i
  - i := i+1



i=5

Depth-first-search

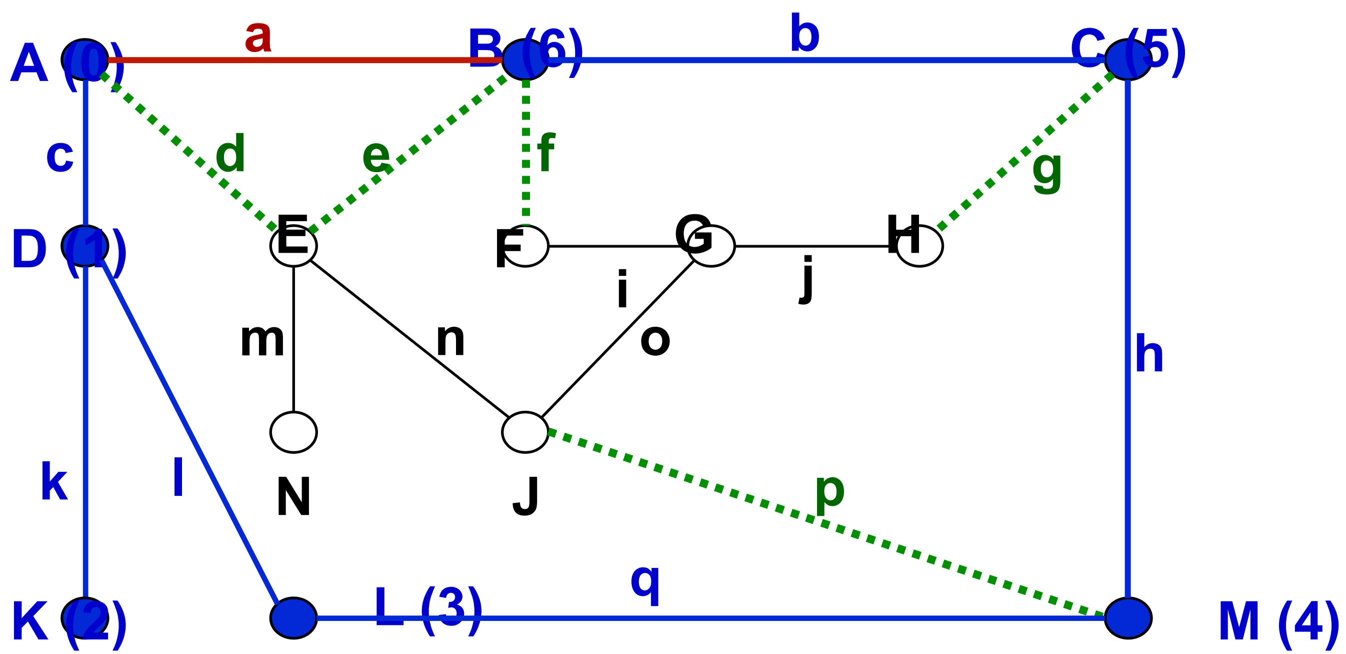
- While T does not span G
  - update the set of frontier edges
  - select a frontier edge whose labelled endpoint has the largest possible dfnumber
  - add this edge to the tree
  - select the unlabelled endpoint of this edge, and set its dfnumber to i
  - i := i+1



i=6

Depth-first-search

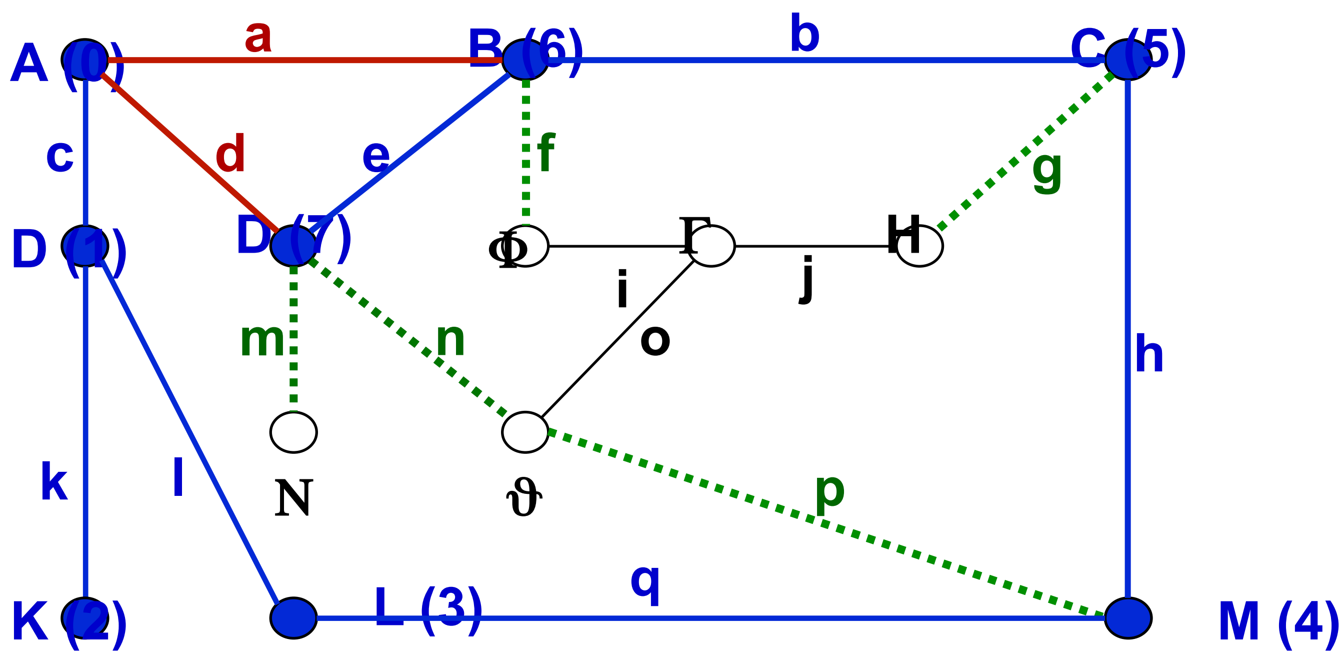
- While T does not span G
  - update the set of frontier edges
  - select a frontier edge whose labelled endpoint has the largest possible dfnumber
  - add this edge to the tree
  - select the unlabelled endpoint of this edge, and set its dfnumber to i
  - i := i+1



i=7

Depth-first-search

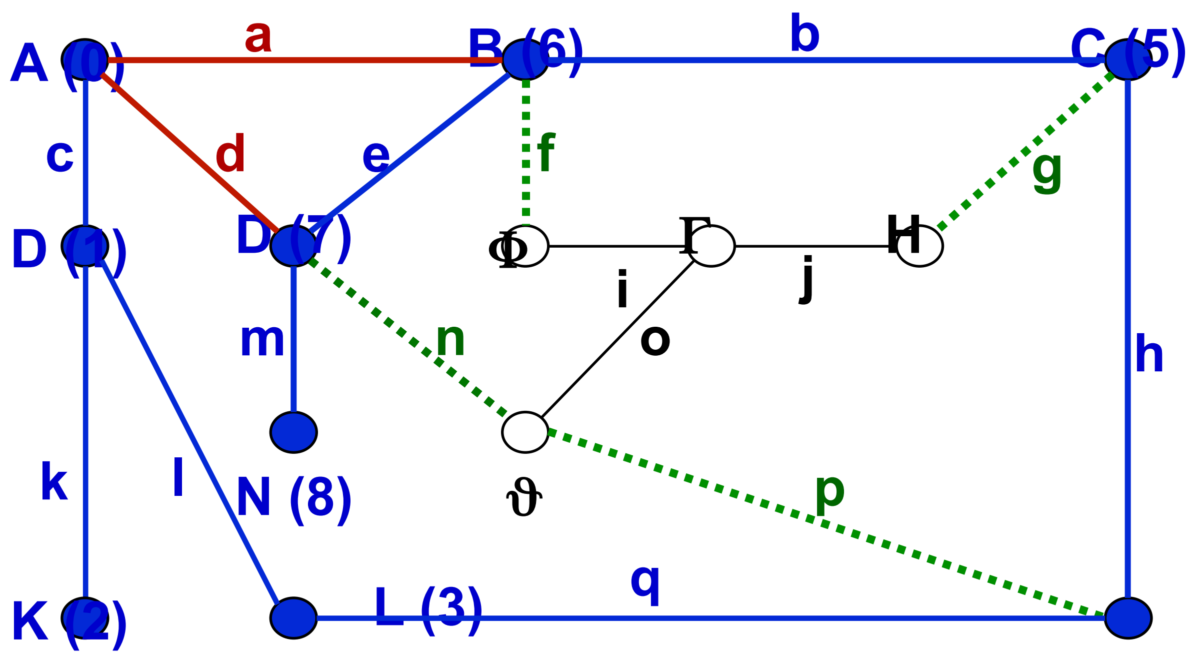
- While T does not span G
  - update the set of frontier edges
  - select a frontier edge whose labelled endpoint has the largest possible dfnumber
  - add this edge to the tree
  - select the unlabelled endpoint of this edge, and set its dfnumber to i
  - i := i+1



i=8

Depth-first-search

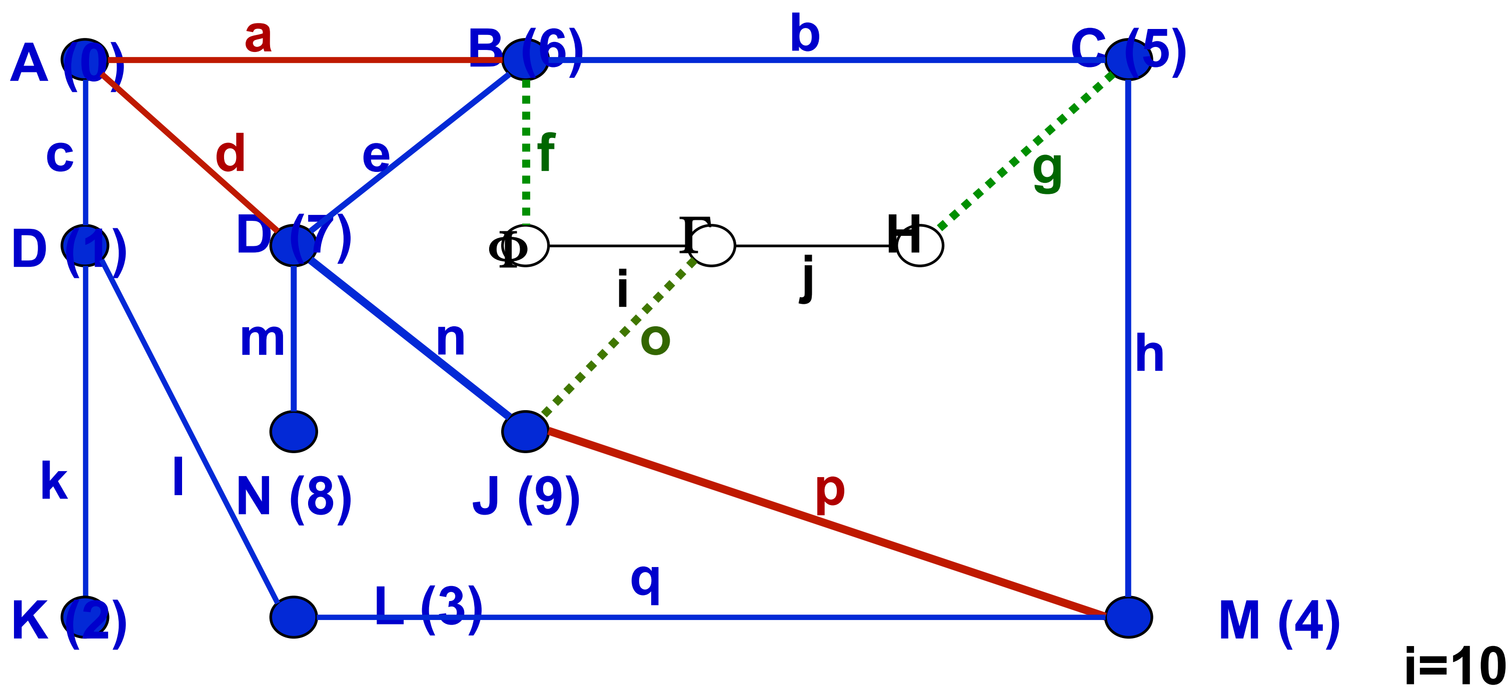
- While T does not span G
  - update the set of frontier edges
  - select a frontier edge whose labelled endpoint has the largest possible dfnumber
  - add this edge to the tree
  - select the unlabelled endpoint of this edge, and set its dfnumber to i
  - i := i+1



i=9

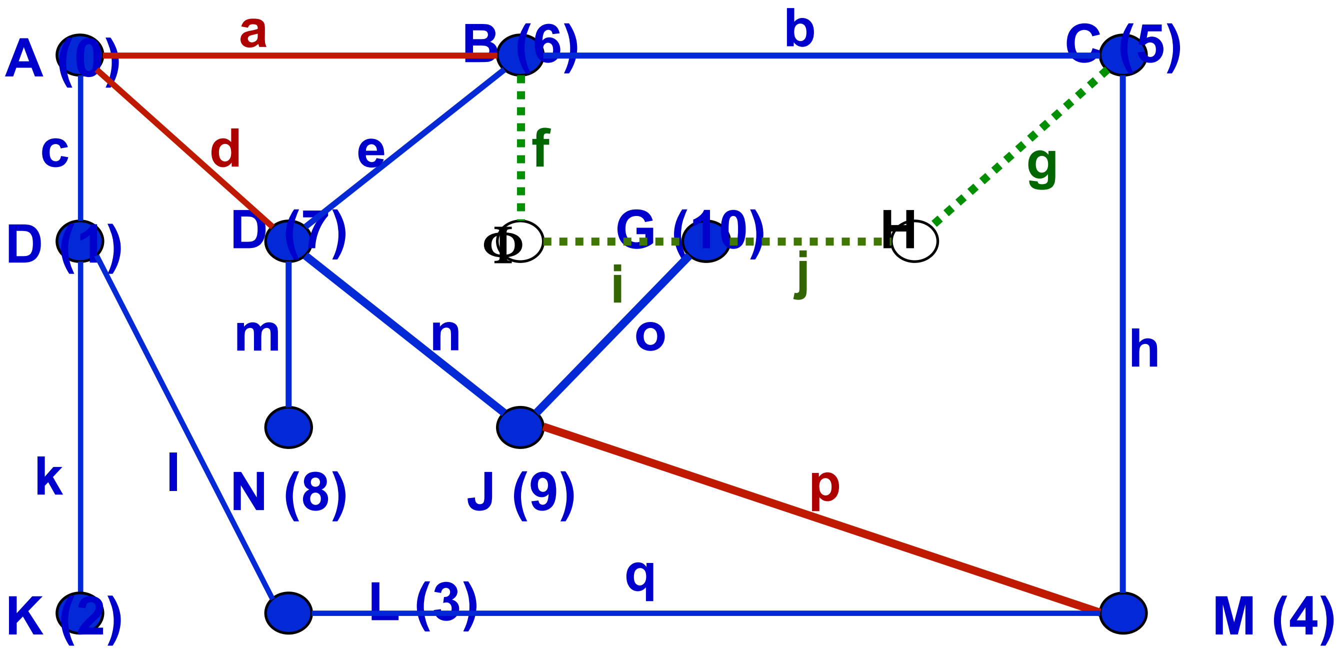
Depth-first-search

- While T does not span G
  - update the set of frontier edges
  - select a frontier edge whose labelled endpoint has the largest possible dfnumber
  - add this edge to the tree
  - select the unlabelled endpoint of this edge, and set its dfnumber to i
  - i := i+1



Depth-first-search

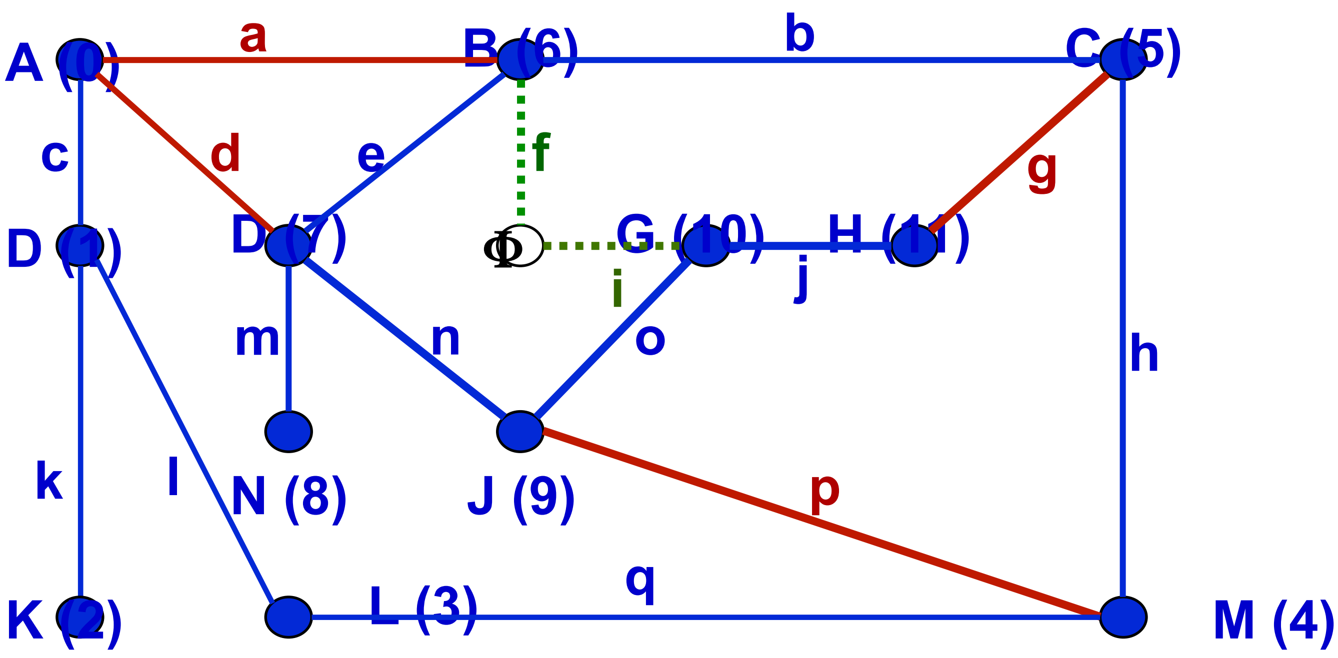
- While T does not span G
  - update the set of frontier edges
  - select a frontier edge whose labelled endpoint has the largest possible dfnumber
  - add this edge to the tree
  - select the unlabelled endpoint of this edge, and set its dfnumber to i
  - i := i+1



i=11

Depth-first-search

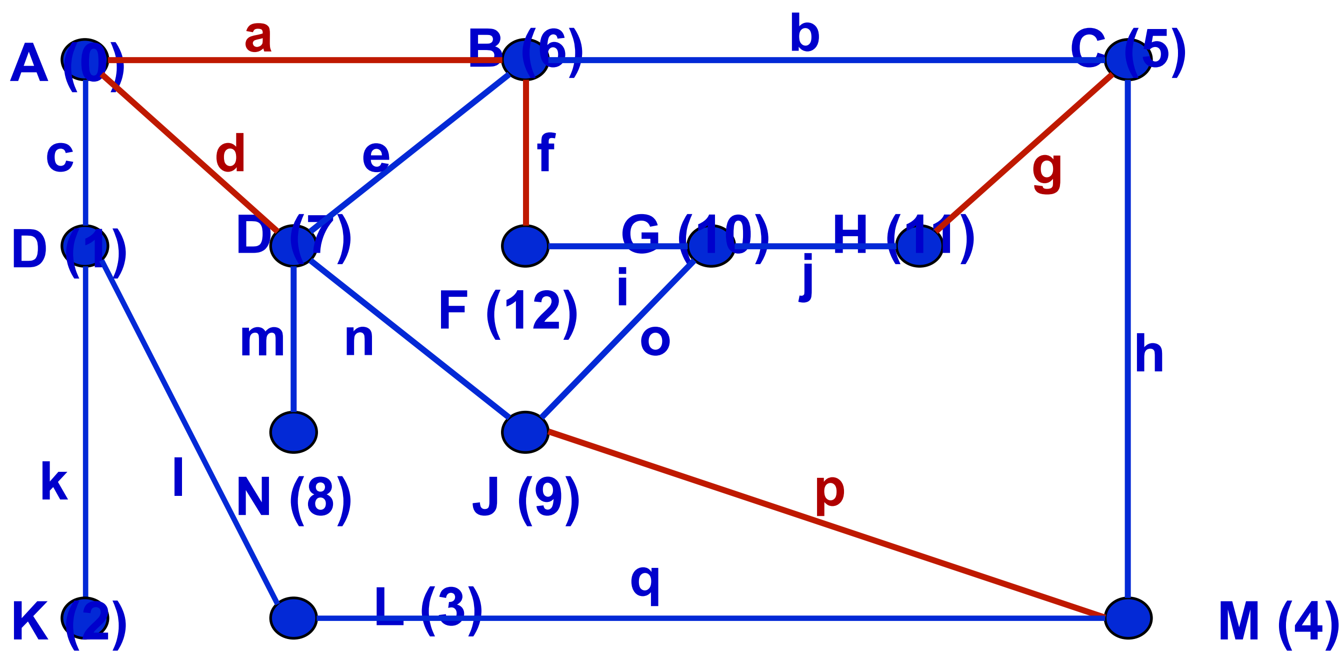
- While T does not span G
  - update the set of frontier edges
  - select a frontier edge whose labelled endpoint has the largest possible dfnumber
  - add this edge to the tree
  - select the unlabelled endpoint of this edge, and set its dfnumber to i
  - i := i+1



i=12

Depth-first-search

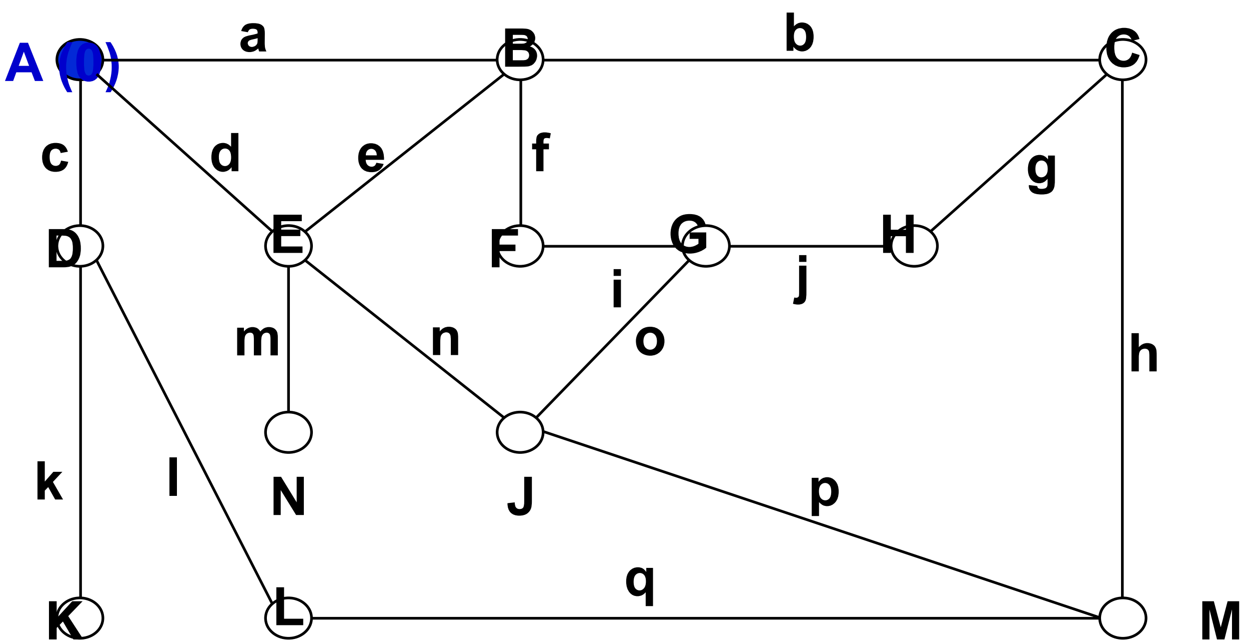
- While T does not span G
  - update the set of frontier edges
  - select a frontier edge whose labelled endpoint has the largest possible dfnumber
  - add this edge to the tree
  - select the unlabelled endpoint of this edge, and set its dfnumber to i
  - i := i+1



i=13

Breadth-first-searcH (BFS)

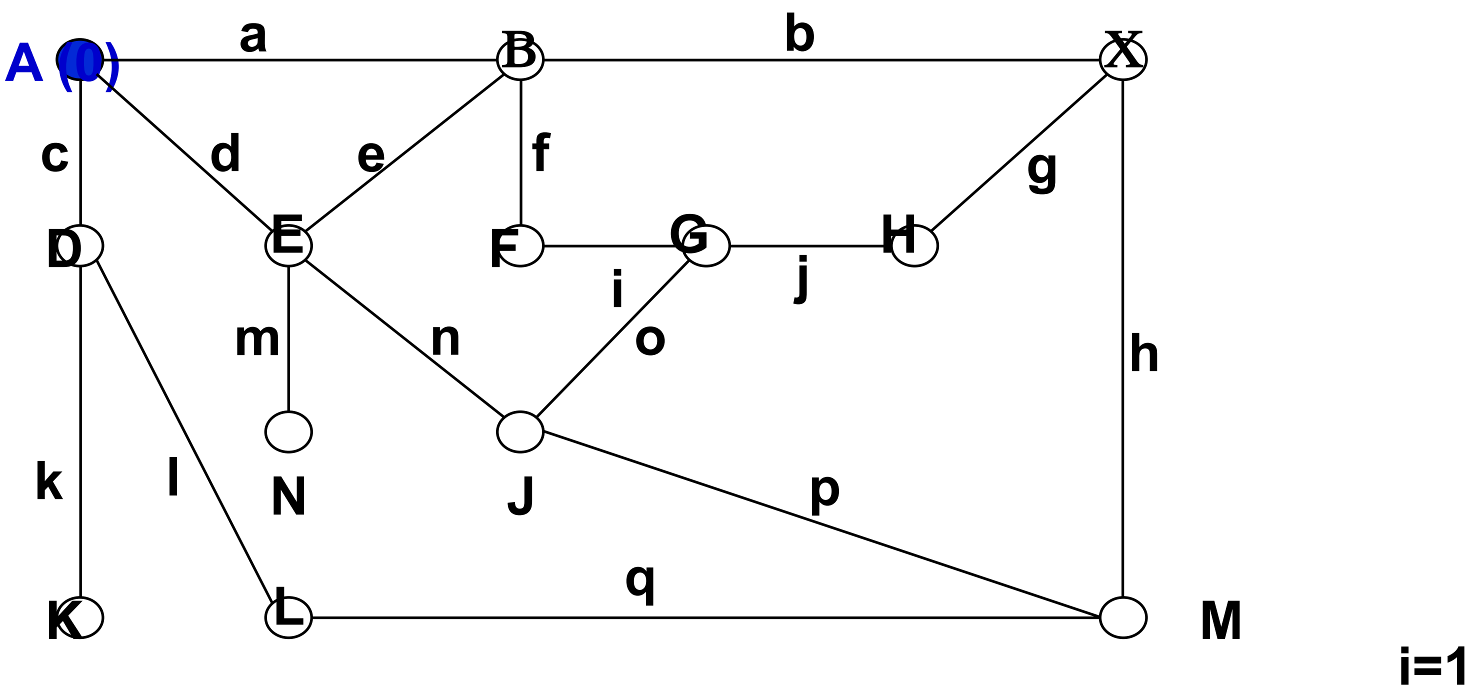
- Initialize tree at a given vertex (for example a)
- Initialize the set of frontier edges as empty
- Write label 0 on vertex a
- Initialize label counter i to 1



i=1

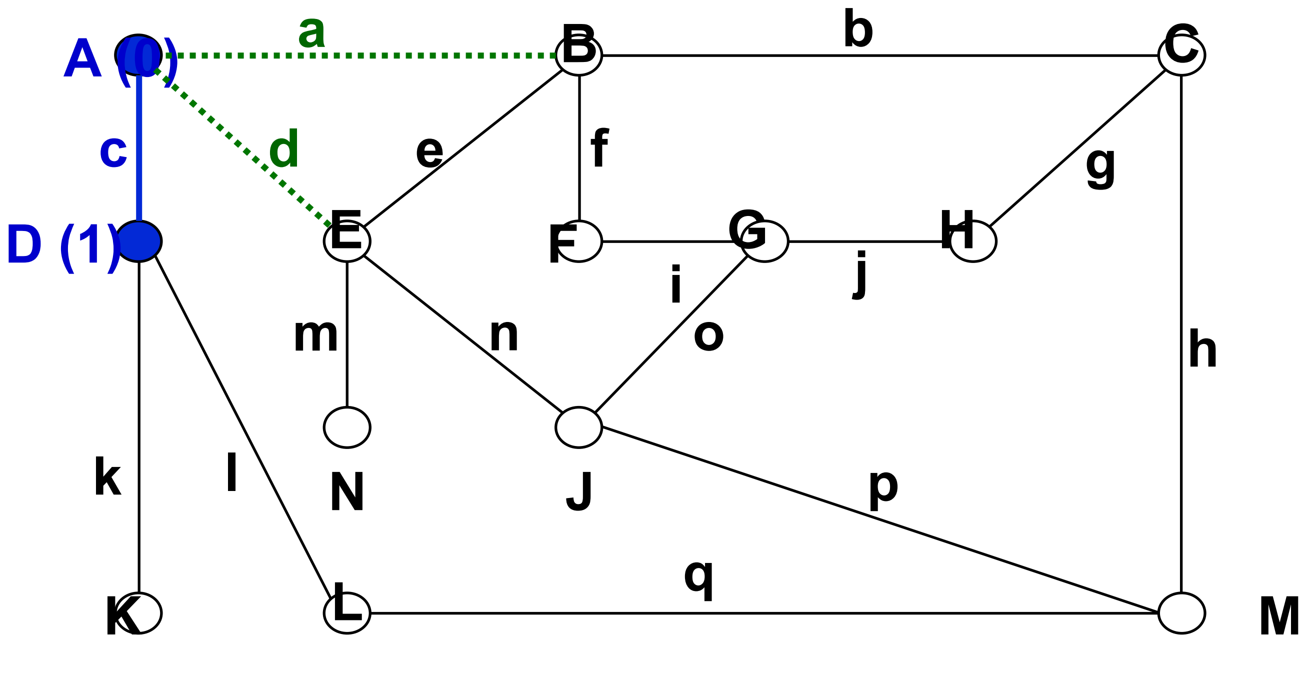
Breadth-first-searcH (BFS)

- While T does not span G
  - update the set of frontier edges
  - select a frontier edge for which the labeled endpoint has the smallest possible label
  - add edge to the tree T
  - select the unlabelled vertex of the edge and set its label to i
  - increment N (i := 1+1)



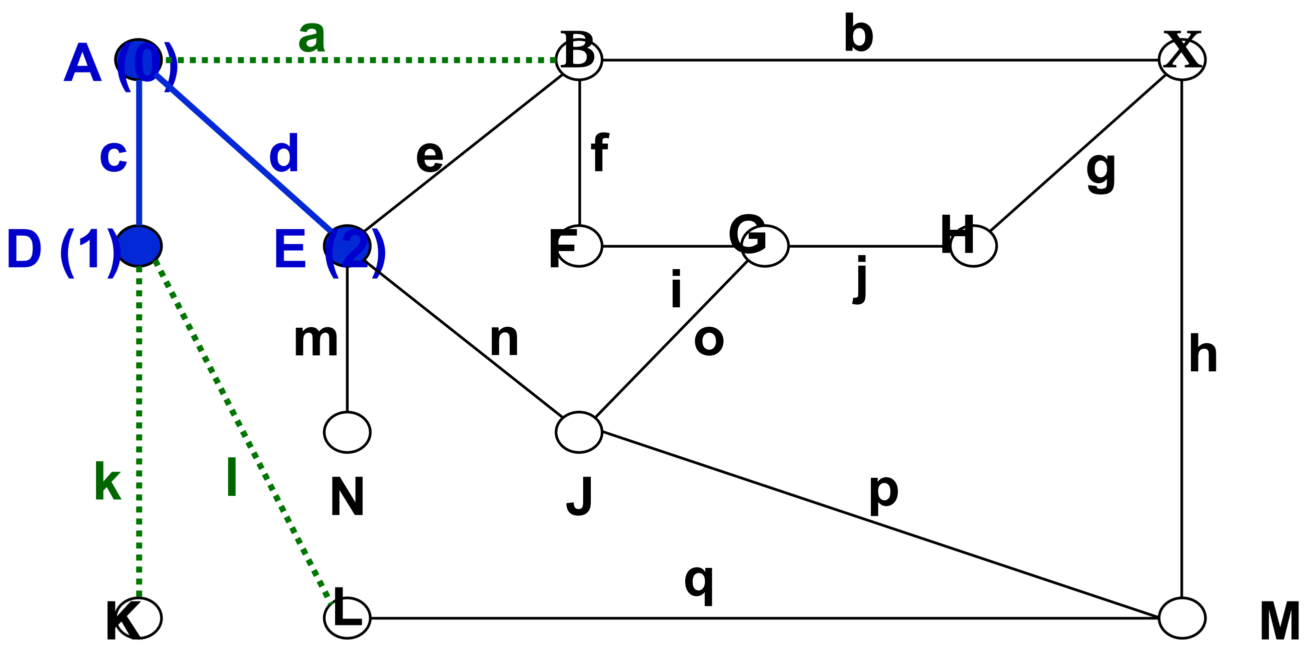
Breadth-first-search

- While T does not span G
  - update the set of frontier edges
  - select a frontier edge for which the labeled endpoint has the smallest possible label
  - add edge to the tree T
  - select the unlabelled vertex of the edge and set its label to i
  - increment N (i := i+1)



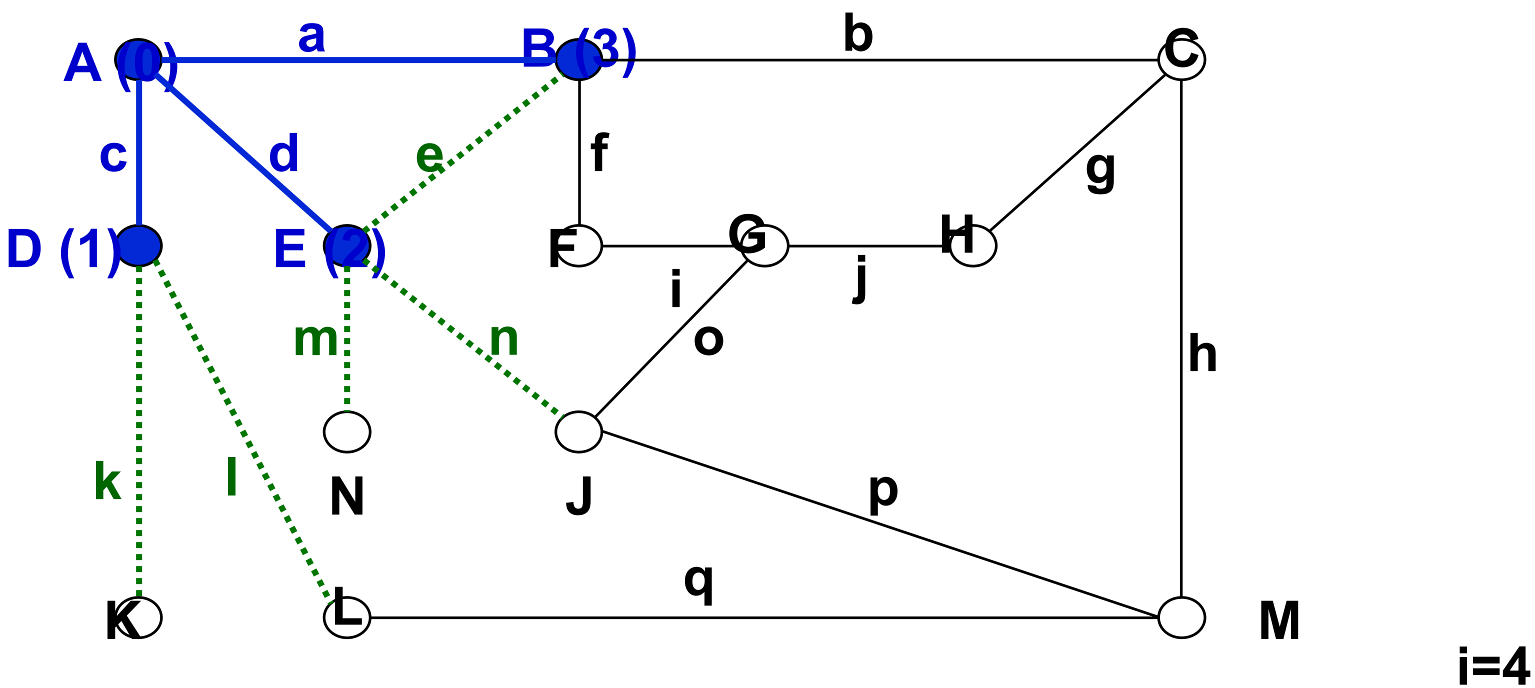
Breadth-first-search

- While T does not span G
  - update the set of frontier edges
  - select a frontier edge for which the labeled endpoint has the smallest possible label
  - add edge to the tree T
  - select the unlabelled vertex of the edge and set its label to i
  - increment N (i := i+1)



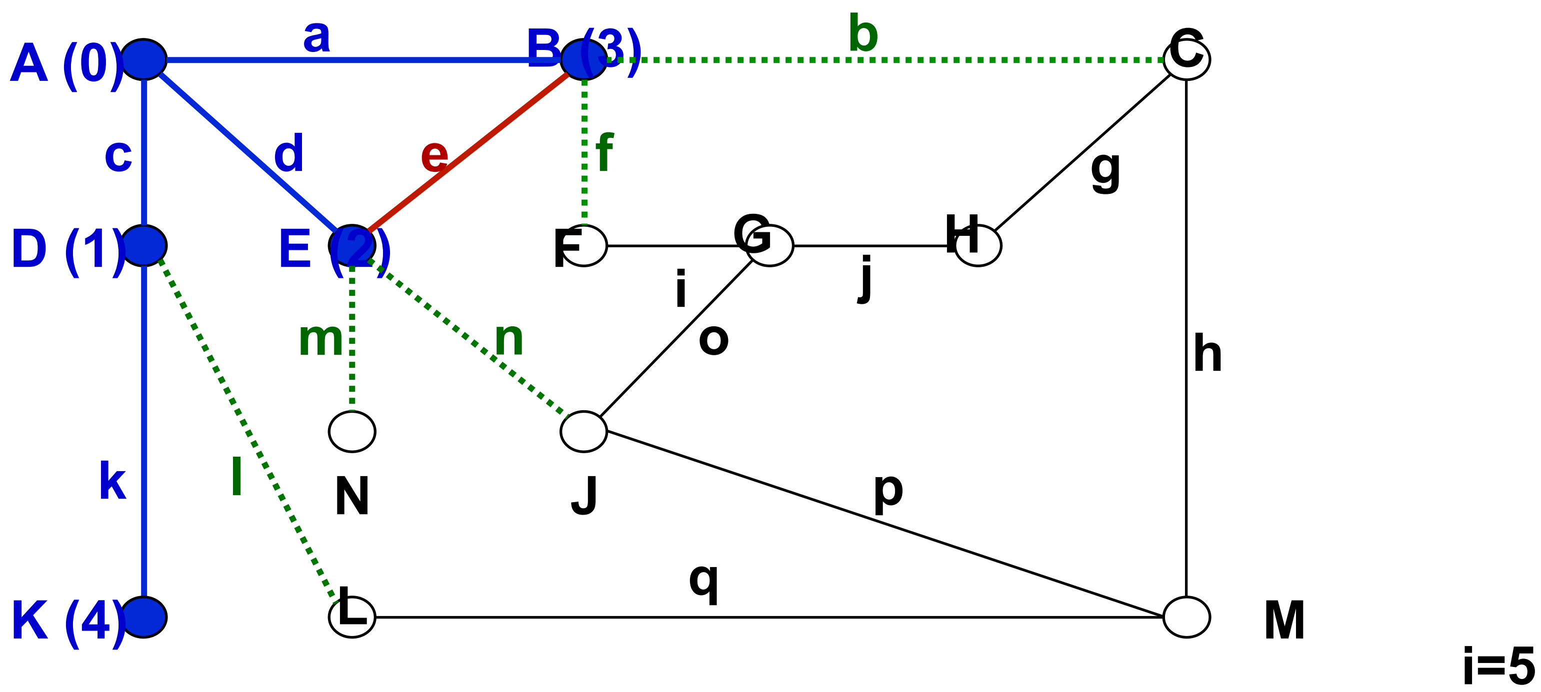
Breadth-first-search

- While T does not span G
  - update the set of frontier edges
  - select a frontier edge for which the labeled endpoint has the smallest possible label
  - add edge to the tree T
  - select the unlabelled vertex of the edge and set its label to i
  - increment N (i := i+1)



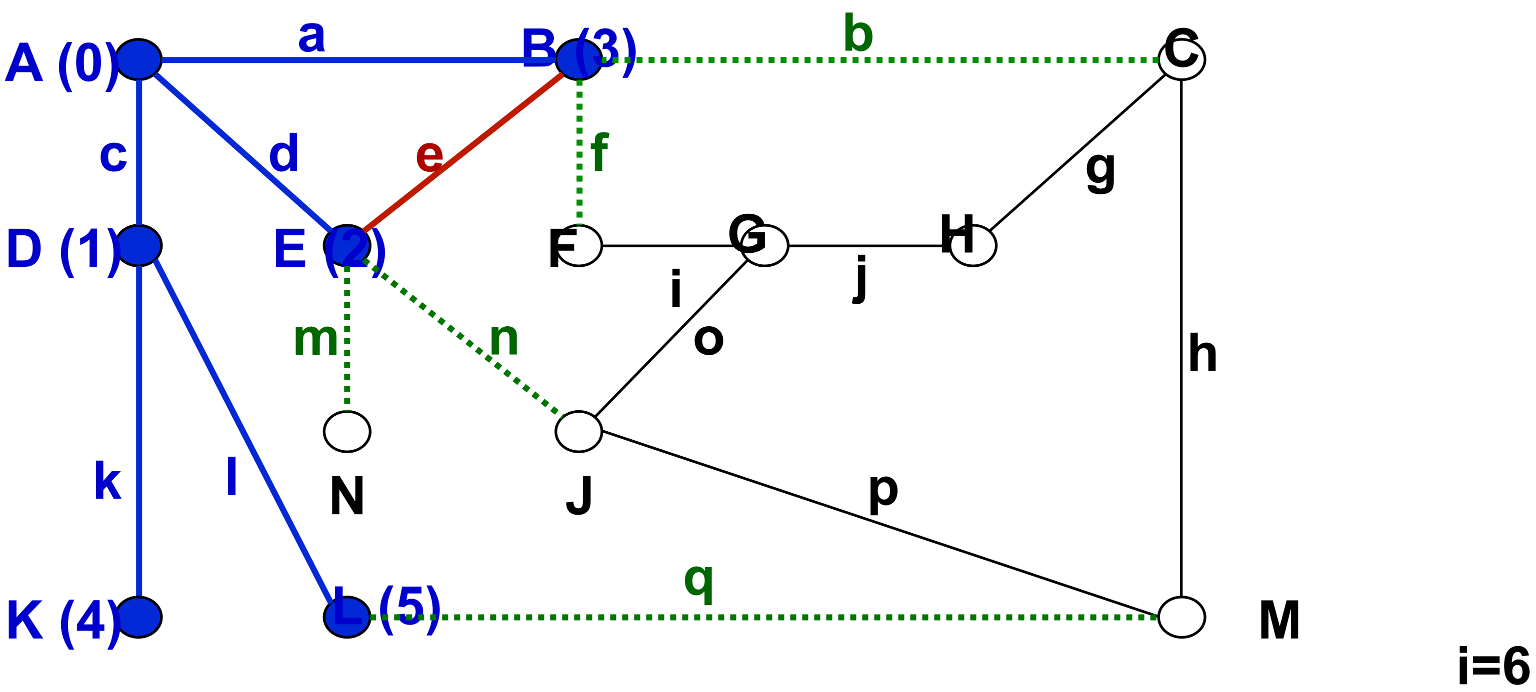
Breadth-first-search

- While T does not span G
  - update the set of frontier edges
  - select a frontier edge for which the labeled endpoint has the smallest possible label
  - add edge to the tree T
  - select the unlabelled vertex of the edge and set its label to i
  - increment N (i := i+1)



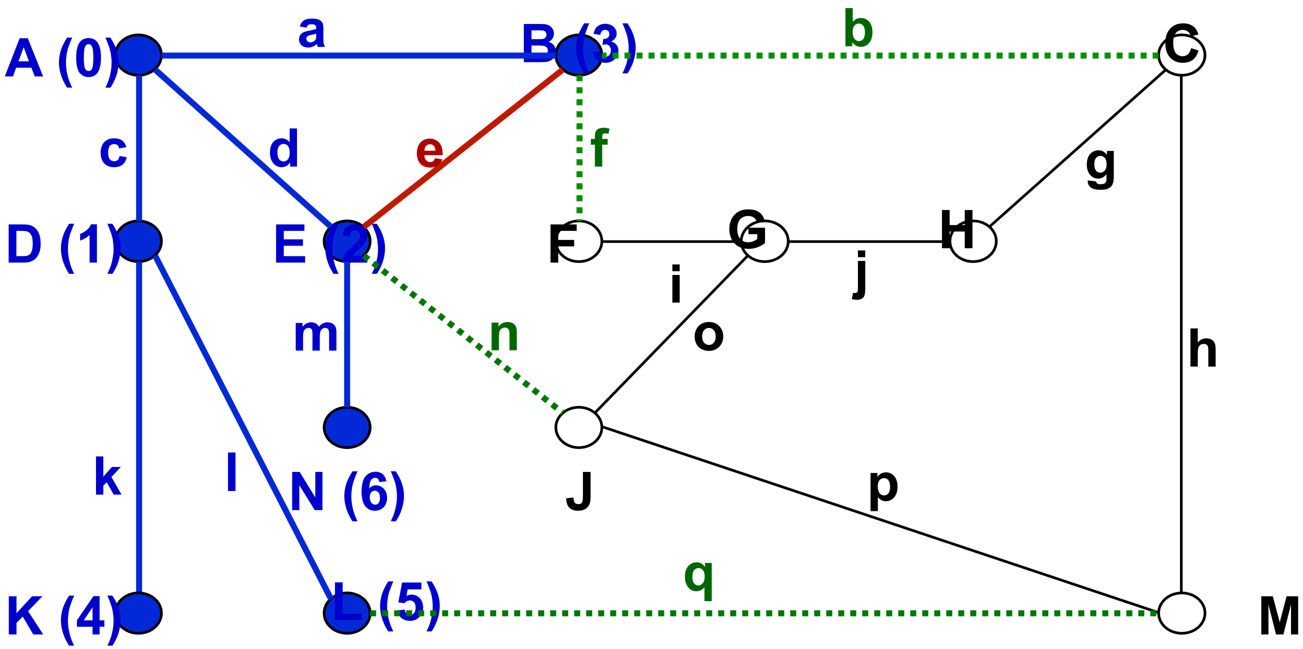
Breadth-first-search

- While T does not span G
  - update the set of frontier edges
  - select a frontier edge for which the labeled endpoint has the smallest possible label
  - add edge to the tree T
  - select the unlabelled vertex of the edge and set its label to i
  - increment N (i := i+1)



Breadth-first-search

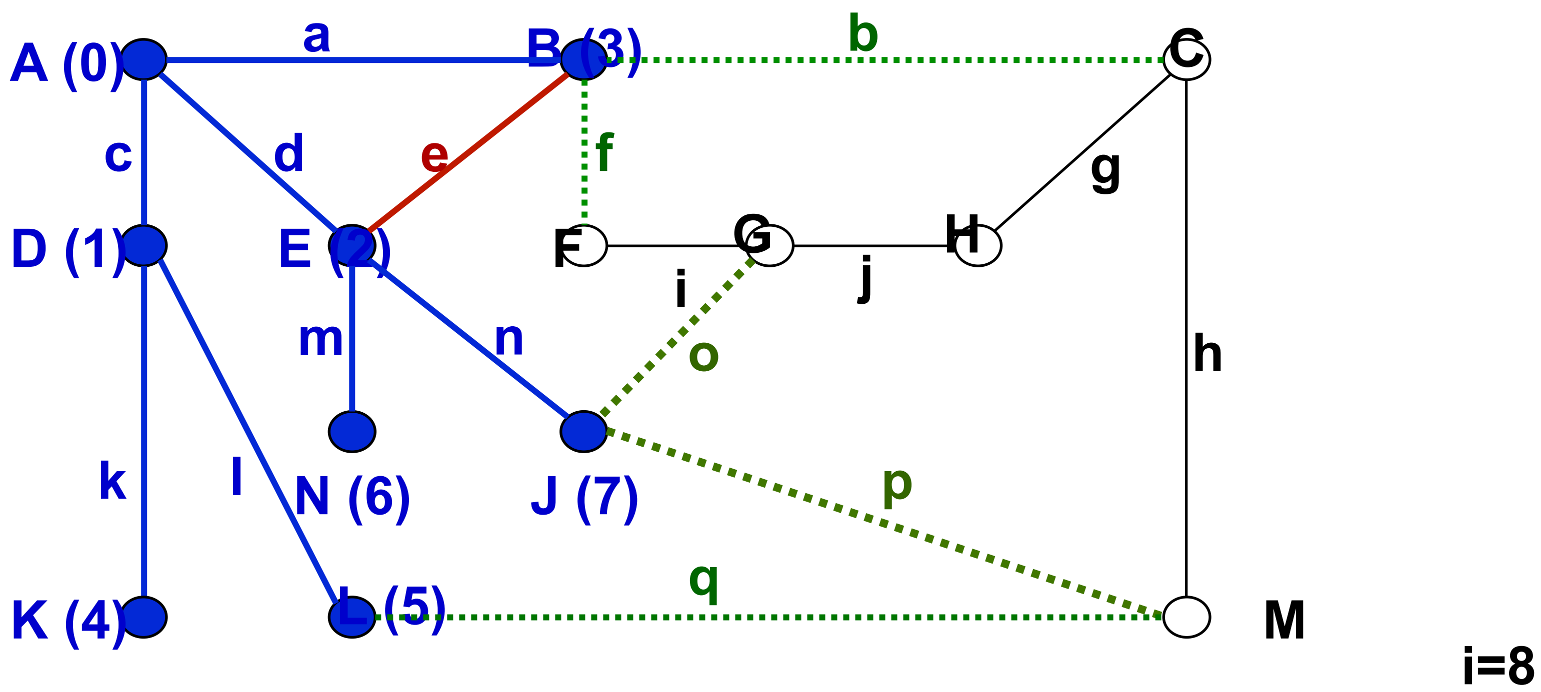
- While T does not span G
  - update the set of frontier edges
  - select a frontier edge for which the labeled endpoint has the smallest possible label
  - add edge to the tree T
  - select the unlabelled vertex of the edge and set its label to i
  - increment N (i := i+1)



i=7

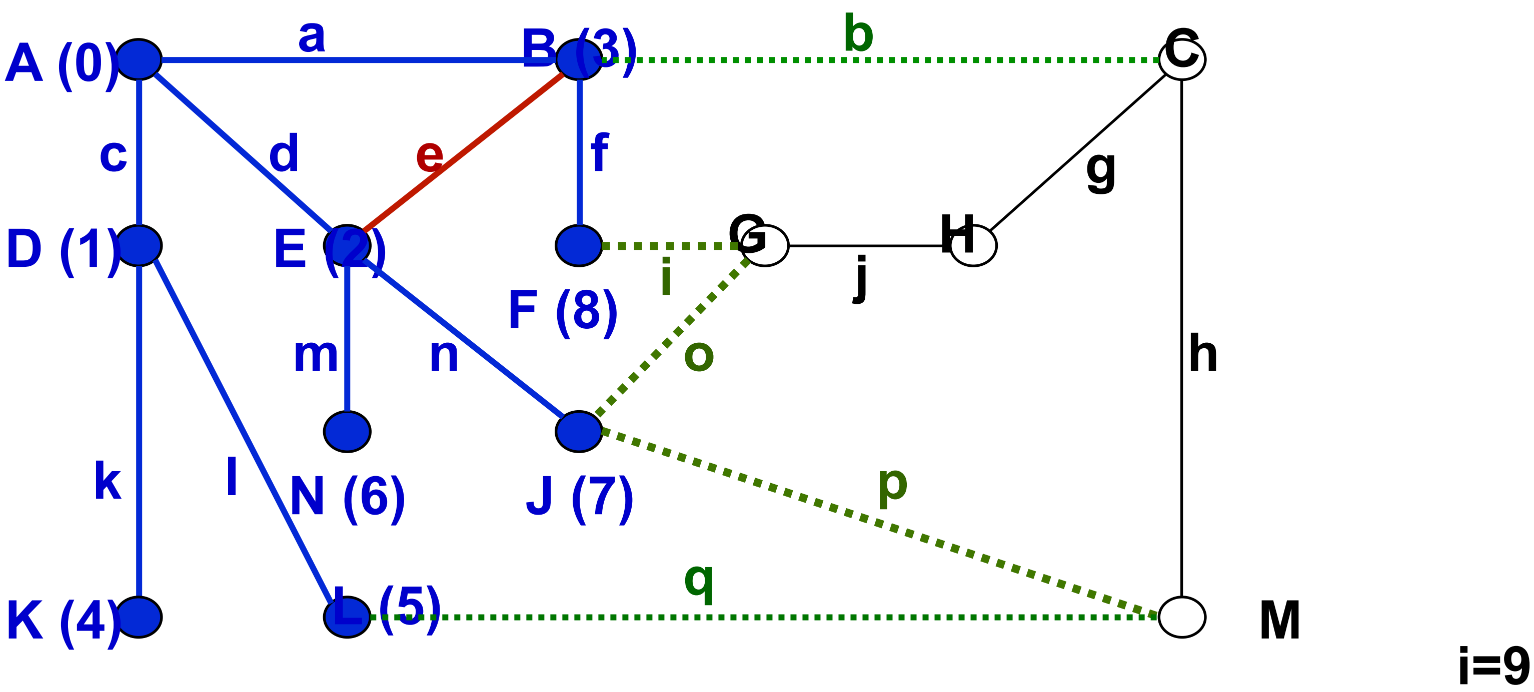
Breadth-first-search

- While T does not span G
  - update the set of frontier edges
  - select a frontier edge for which the labeled endpoint has the smallest possible label
  - add edge to the tree T
  - select the unlabelled vertex of the edge and set its label to i
  - increment N (i := i+1)



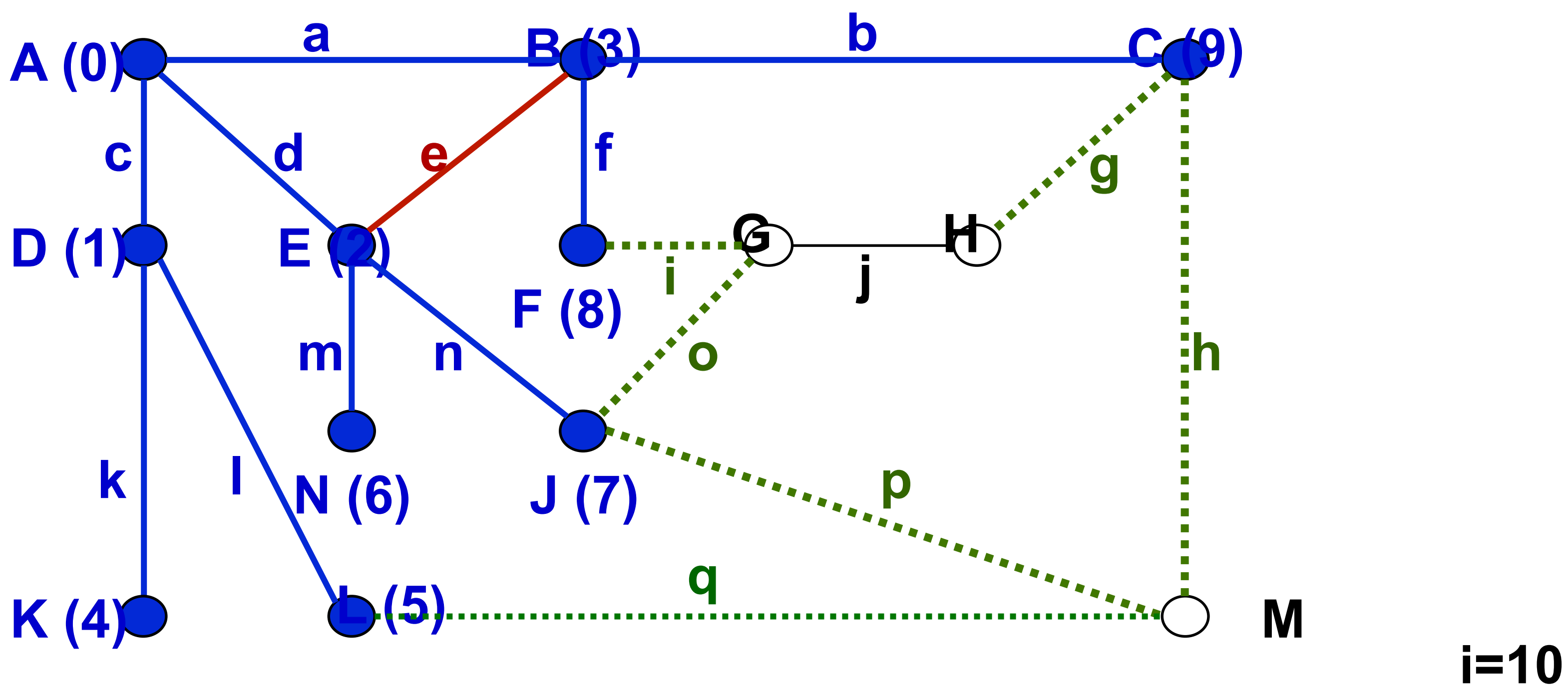
Breadth-first-search

- While T does not span G
  - update the set of frontier edges
  - select a frontier edge for which the labeled endpoint has the smallest possible label
  - add edge to the tree T
  - select the unlabelled vertex of the edge and set its label to i
  - increment N (i := i+1)



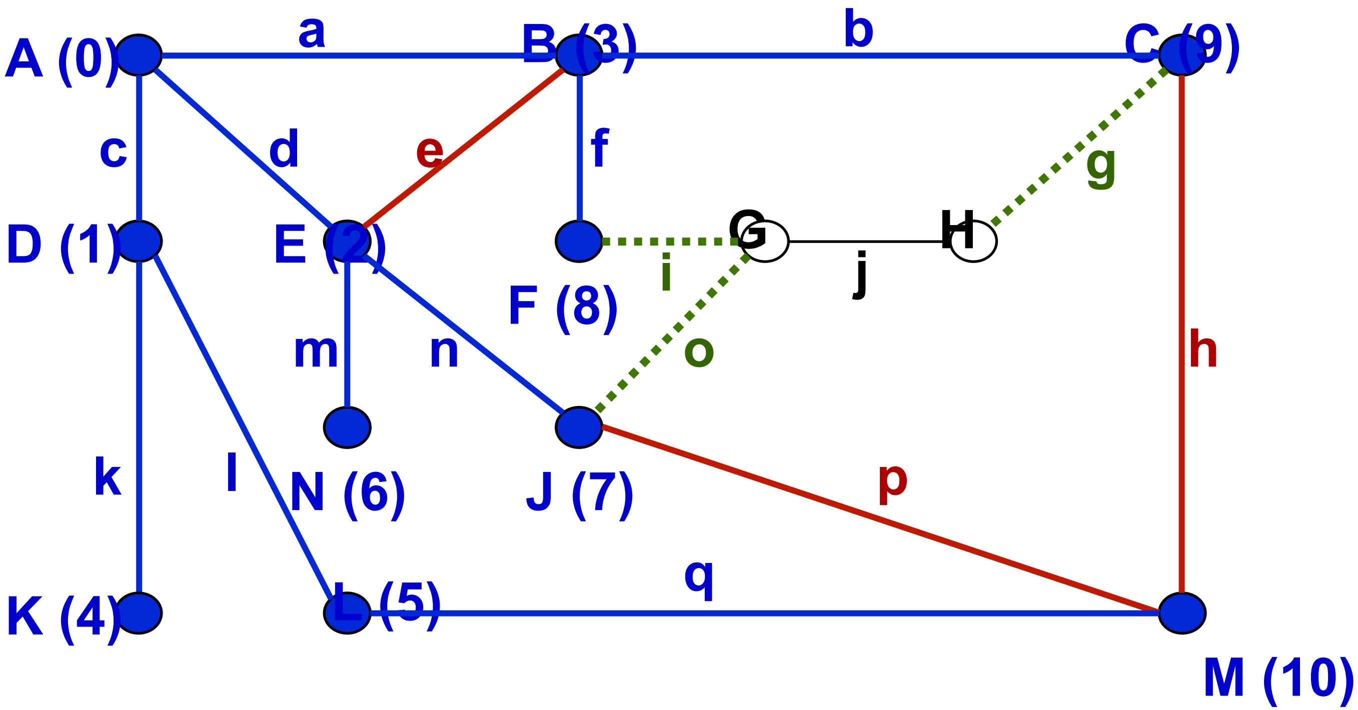
Breadth-first-search

- While T does not span G
  - update the set of frontier edges
  - select a frontier edge for which the labeled endpoint has the smallest possible label
  - add edge to the tree T
  - select the unlabelled vertex of the edge and set its label to i
  - increment N (i := i+1)



Breadth-first-search

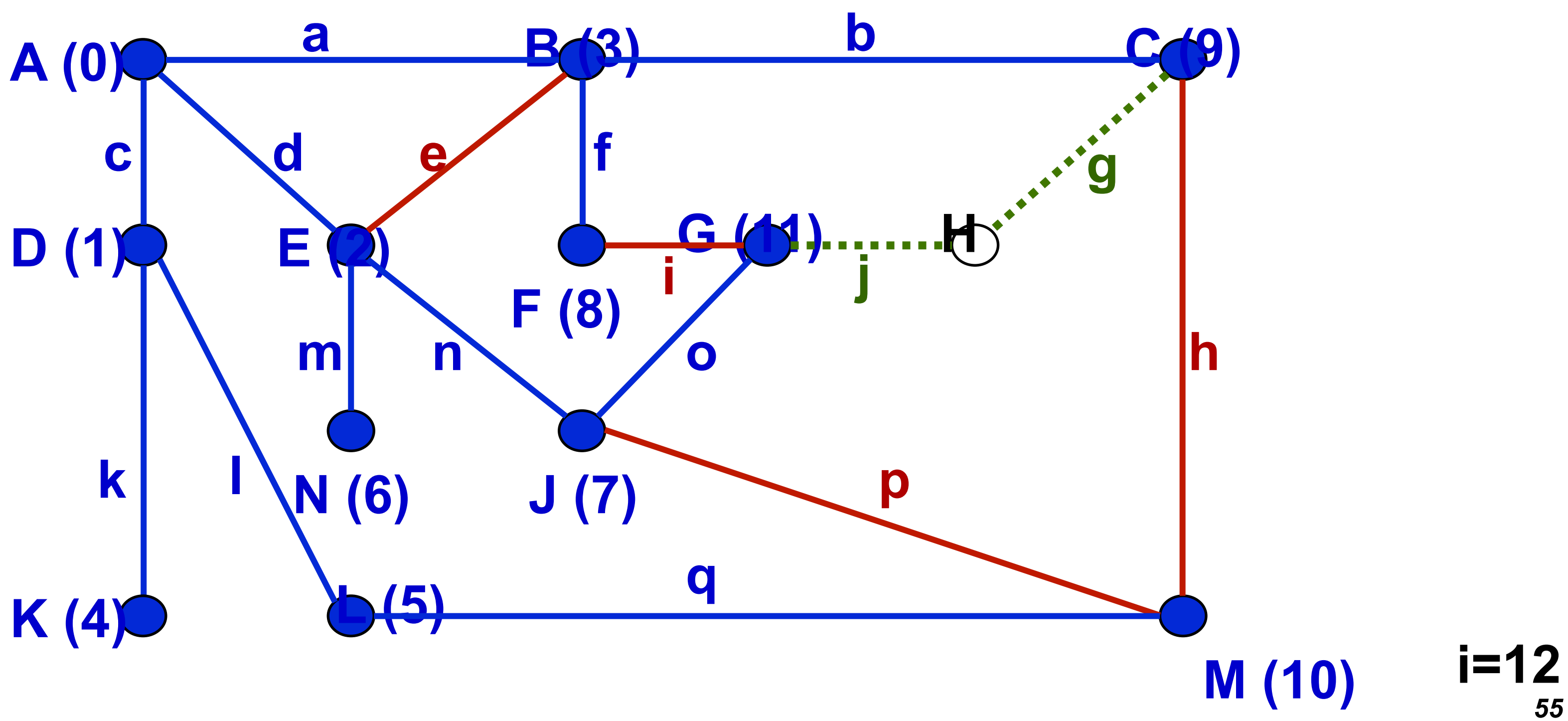
- While T does not span G
  - update the set of frontier edges
  - select a frontier edge for which the labeled endpoint has the smallest possible label
  - add edge to the tree T
  - select the unlabelled vertex of the edge and set its label to i
  - increment N (i := i+1)



i=11

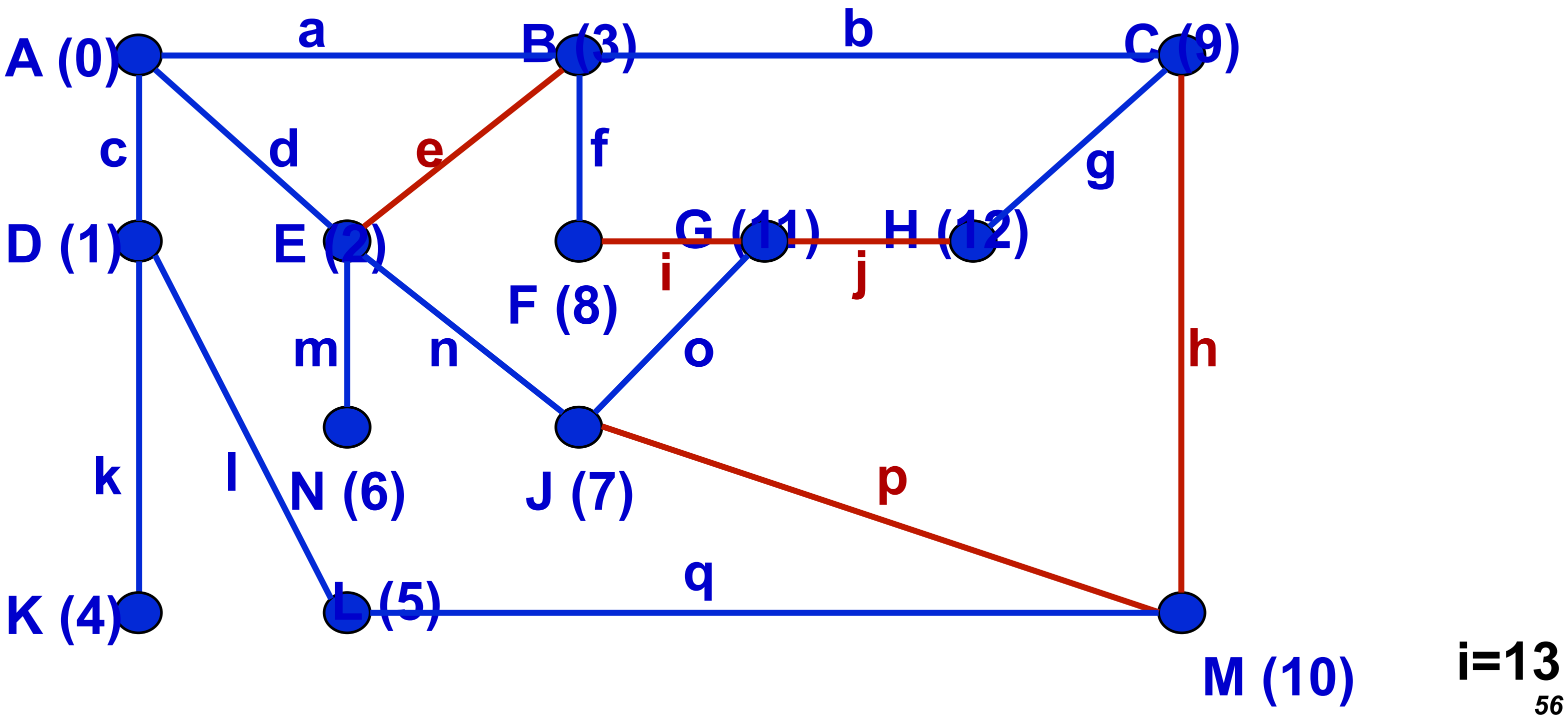
Breadth-first-search

- While T does not span G
  - update the set of frontier edges
  - select a frontier edge for which the labeled endpoint has the smallest possible label
  - add edge to the tree T
  - select the unlabelled vertex of the edge and set its label to i
  - increment N (i := i+1)

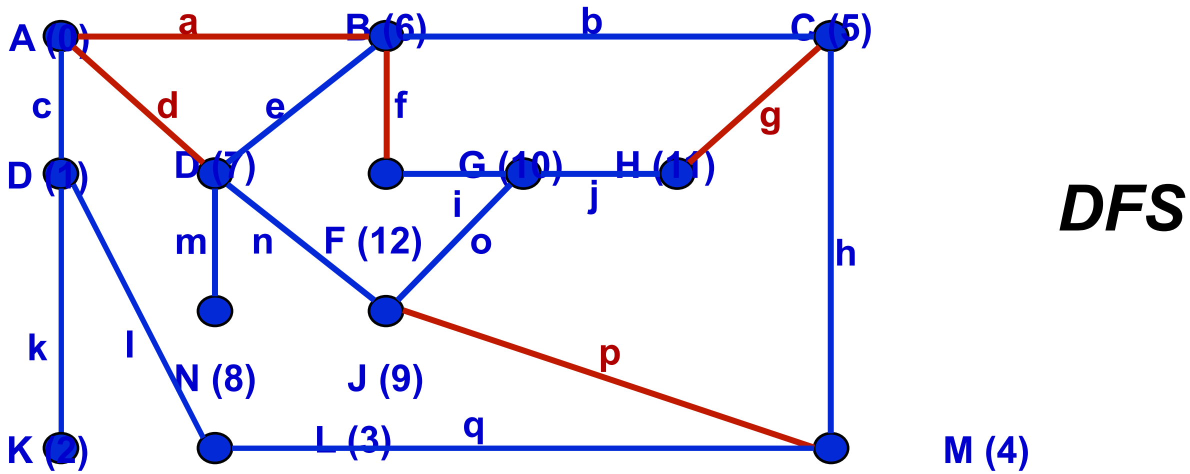


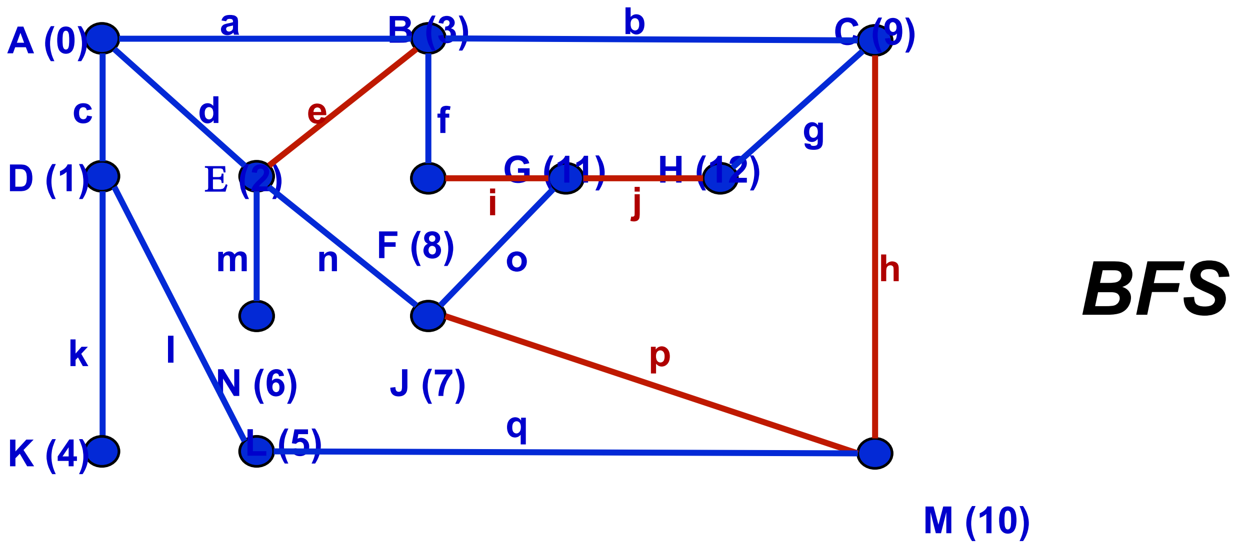
Breadth-first-search

- While T does not span G
  - update the set of frontier edges
  - select a frontier edge for which the labeled endpoint has the smallest possible label
  - add edge to the tree T
  - select the unlabelled vertex of the edge and set its label to i
  - increment N (i := i+1)



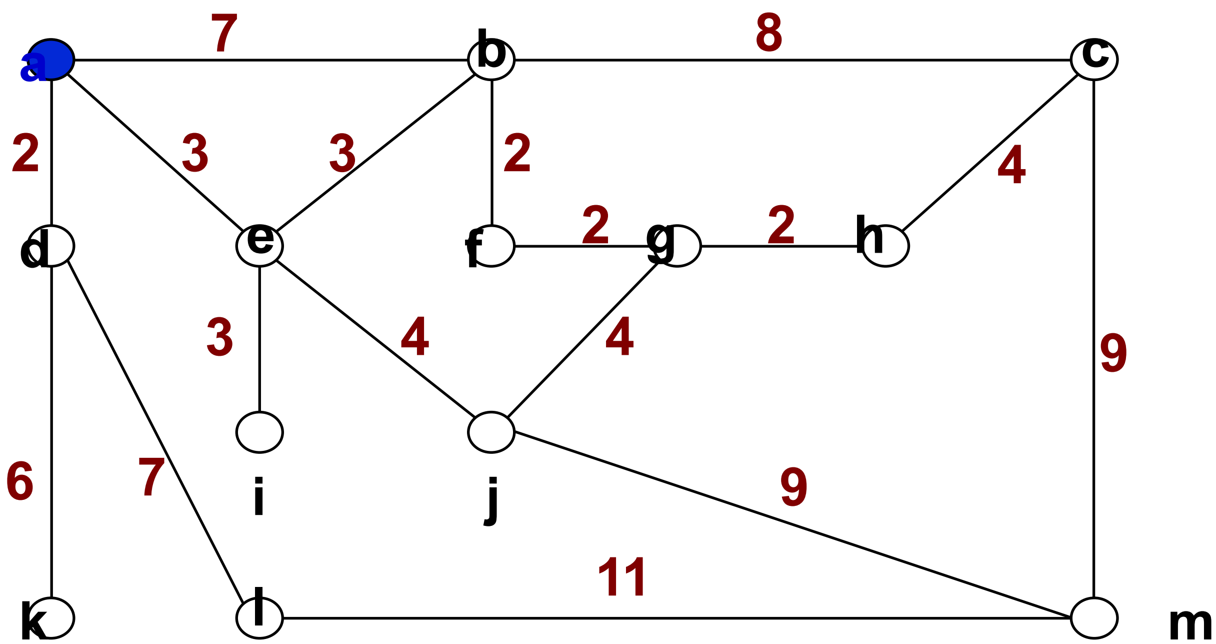
Result comparison : DFS versus BFS





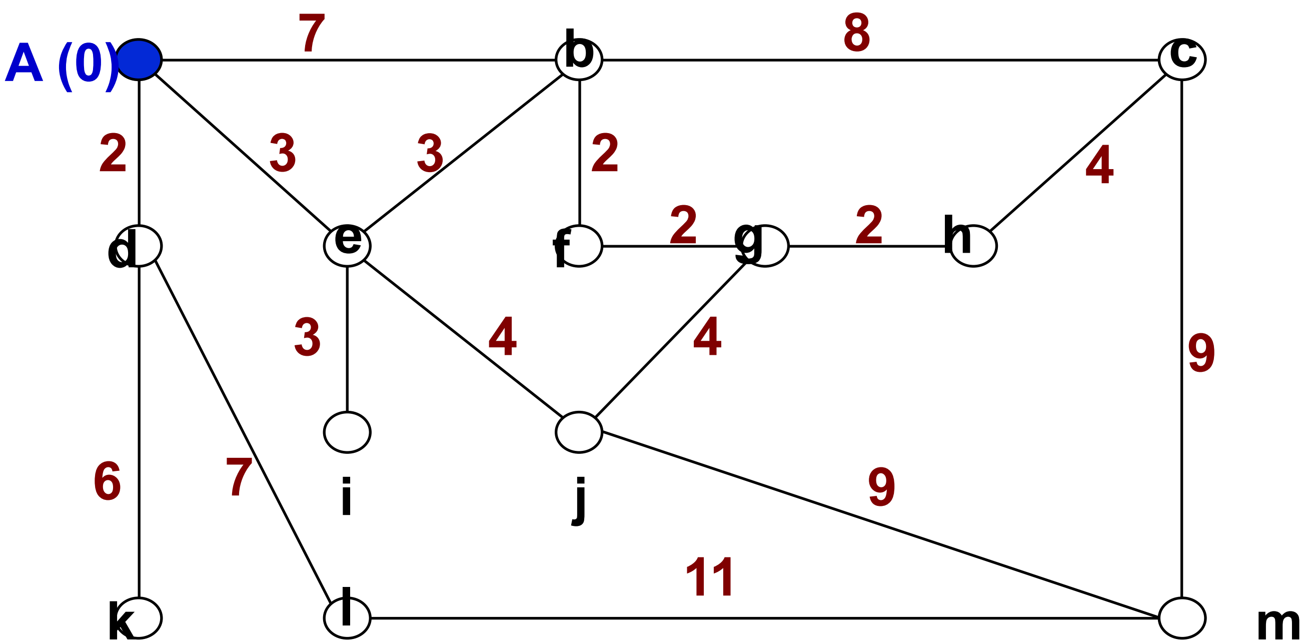
Dijkstra's Shortest path finding algorithm

- Input: a weighted connected graph G whose edge-weights are non-negative, and a starting vertex **a**
- Output: a spanning tree, rooted at **a**, whose path from each vertex v is the shortest path from **a** to v in G; the vertex-labelling gives the distance from s to each vertex



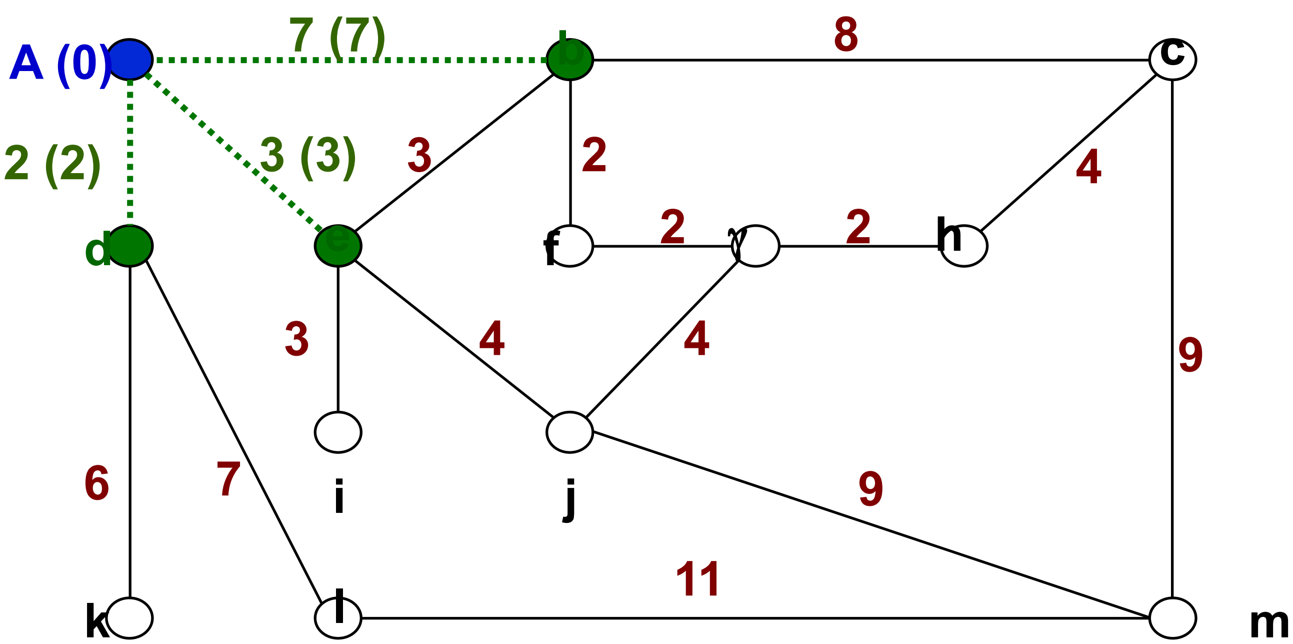
Dijkstra's Shortest path finding algorithm

- Initialise the Dijkstra tree T as vertex **a**
- **dist[a]** = 0
- write label 0 on vertex **a**



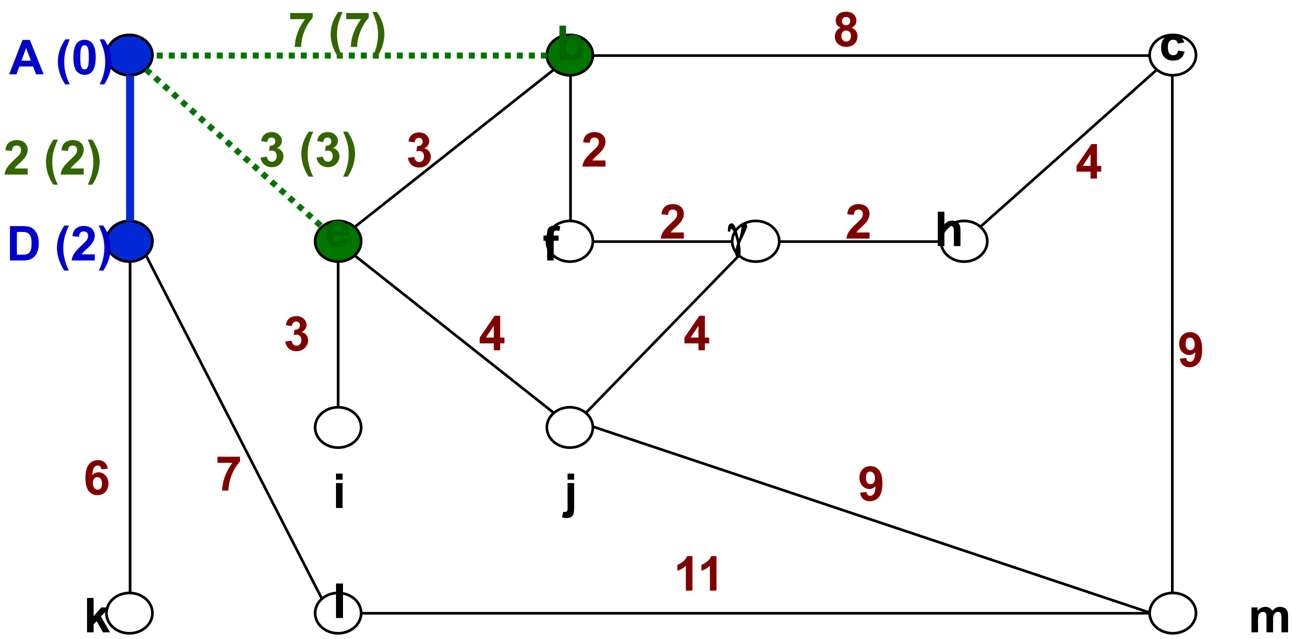
Dijkstra's Shortest path finding algorithm

- While T does not span G
  - Update frontier edges
  - For each frontier edge **e**
    - let **x** be the labelled and **y** the unlabelled endpoints of e
    - set **D(e)** = dist[x] + w(e)



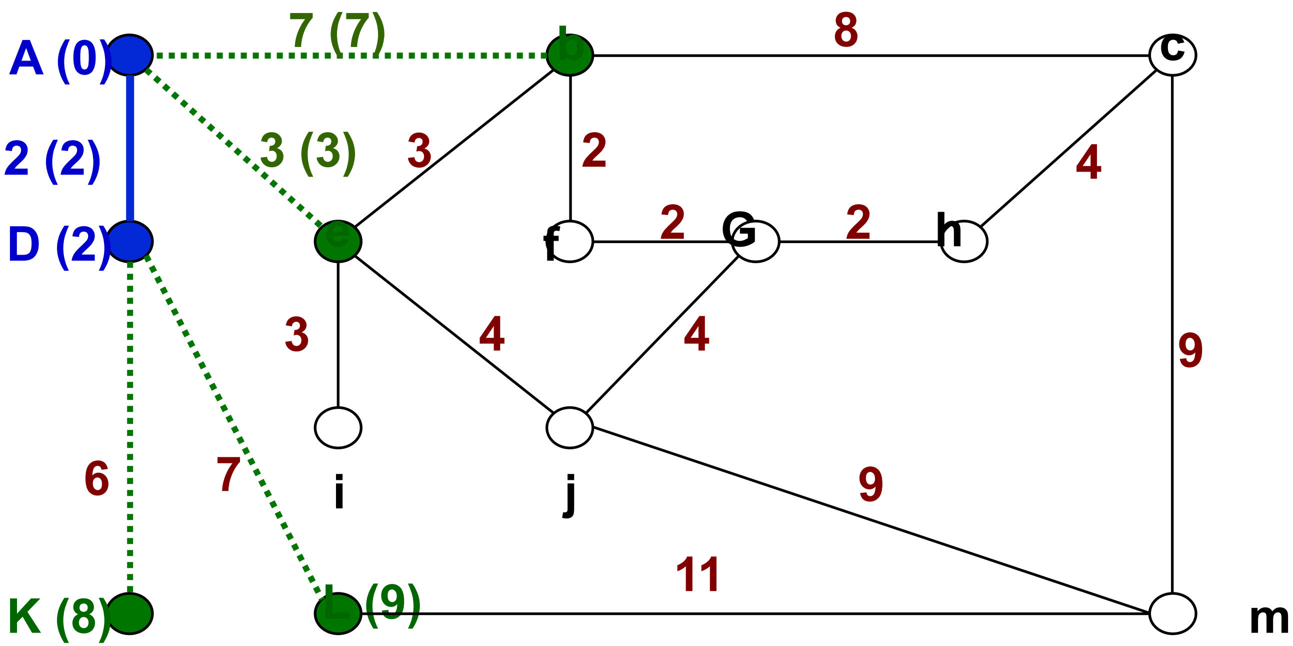
Dijkstra's Shortest path finding algorithm

- While T does not span G
  - Update frontier edges
  - For each frontier edge **e**
    - let **x** be the labelled and **y** the unlabelled endpoints of e
    - set **D(e)** = dist[x] + w(e)
  - Let **e** be a frontier edge that has the smallest D-value
  - Add edge e to T, and set **dist[y]** = D(e)



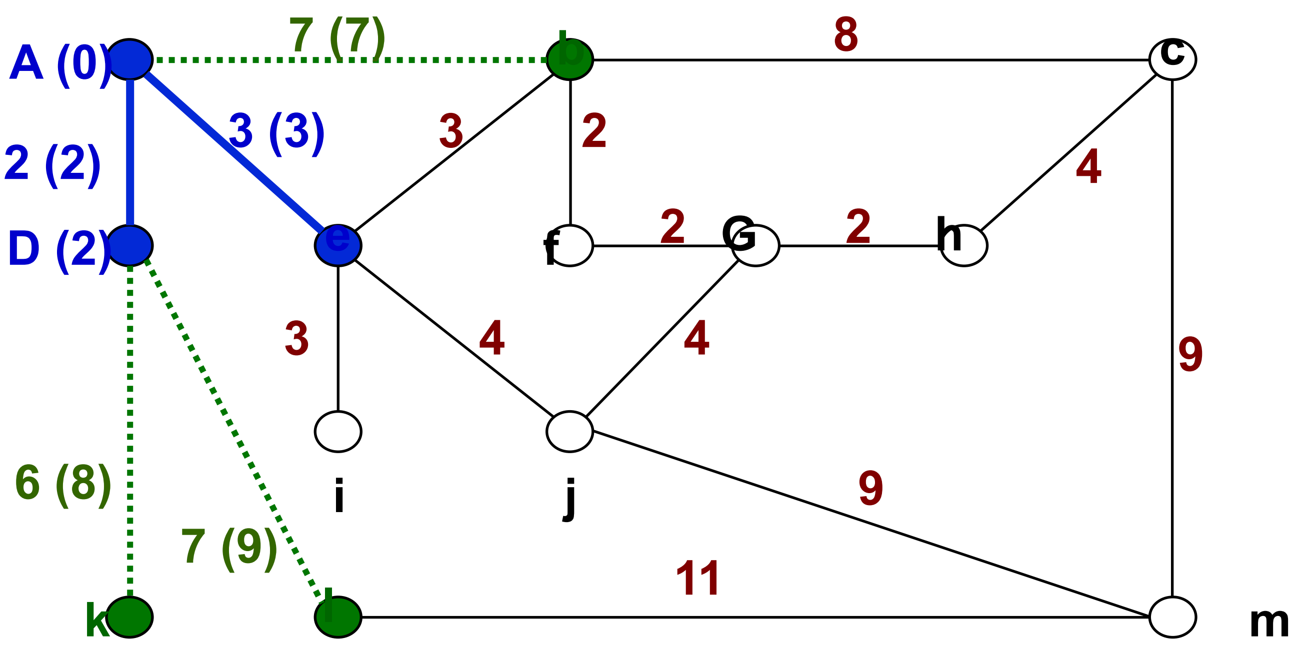
Dijkstra's Shortest path finding algorithm

- While T does not span G
  - Update frontier edges
  - For each frontier edge **e**
    - let **x** be the labelled and **y** the unlabelled endpoints of e
    - set **D(e)** = dist[x] + w(e)
  - Let **e** be a frontier edge that has the smallest D-value
  - Add edge e to T, and set **dist[y]** = D(e)



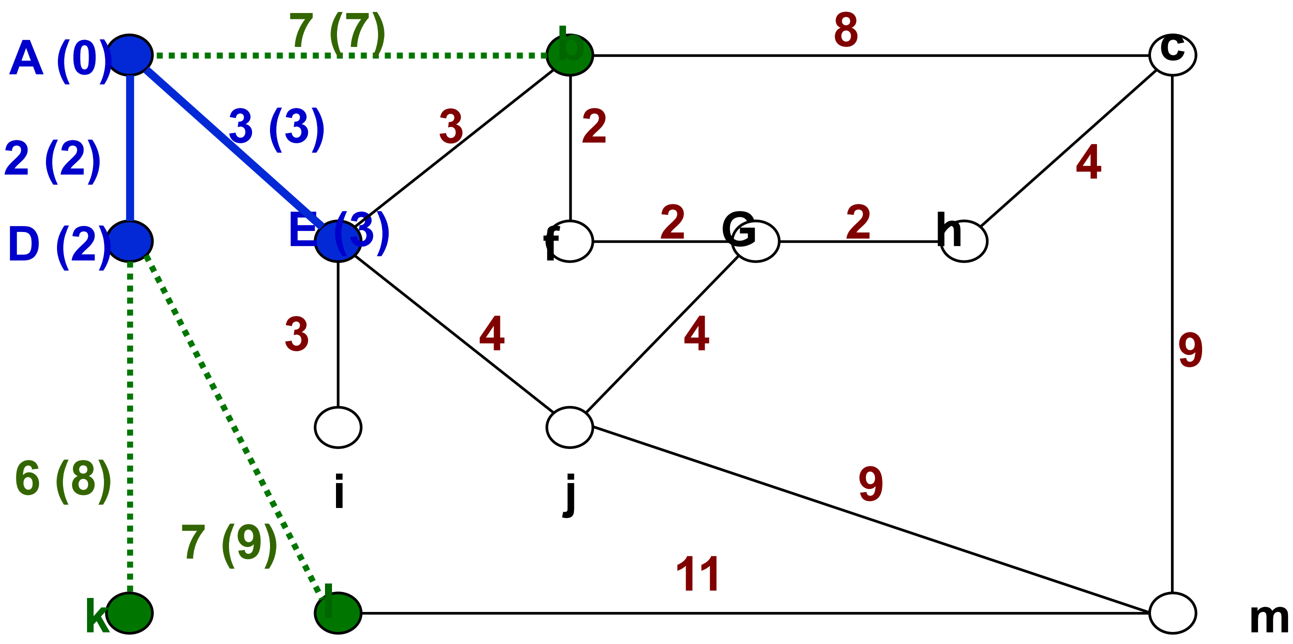
Dijkstra's Shortest path finding algorithm

- While T does not span G
  - Update frontier edges
  - For each frontier edge **e**
    - let **x** be the labelled and **y** the unlabelled endpoints of e
    - set **D(e)** = dist[x] + w(e)
  - Let **e** be a frontier edge that has the smallest D-value
  - Add edge e to T, and set **dist[y]** = D(e)



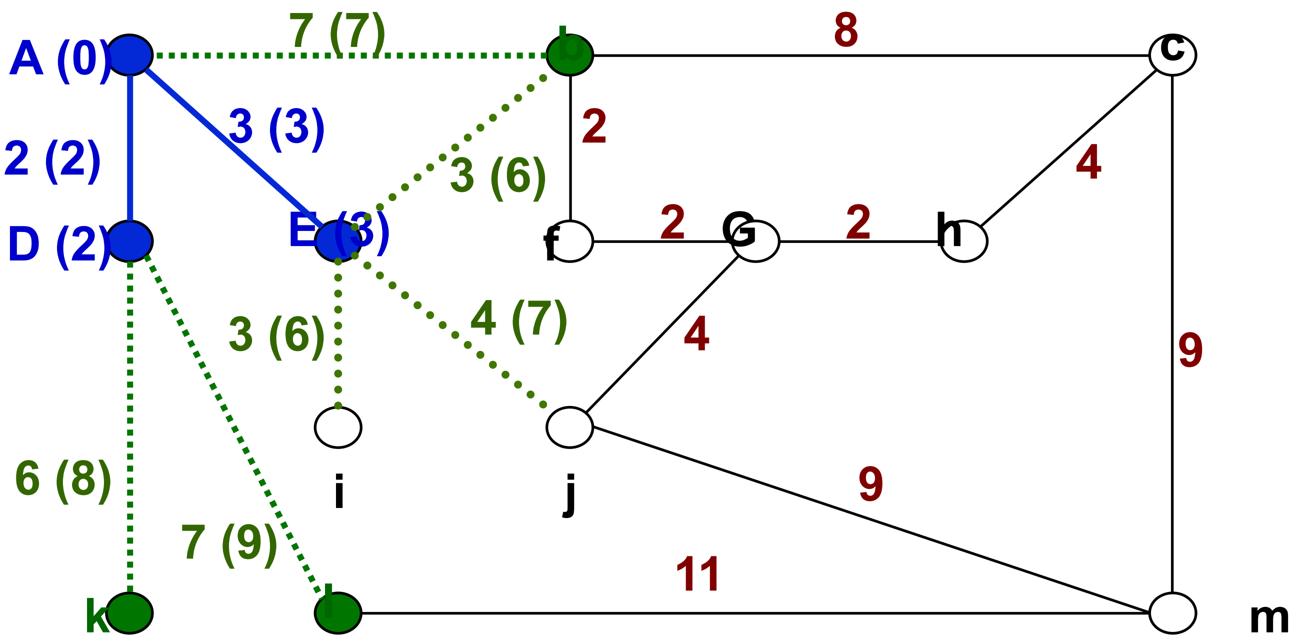
Dijkstra's Shortest path finding algorithm

- While T does not span G
  - Update frontier edges
  - For each frontier edge **e**
    - let **x** be the labelled and **y** the unlabelled endpoints of e
    - set **D(e)** = dist[x] + w(e)
  - Let **e** be a frontier edge that has the smallest D-value
  - Add edge e to T, and set **dist[y]** = D(e)



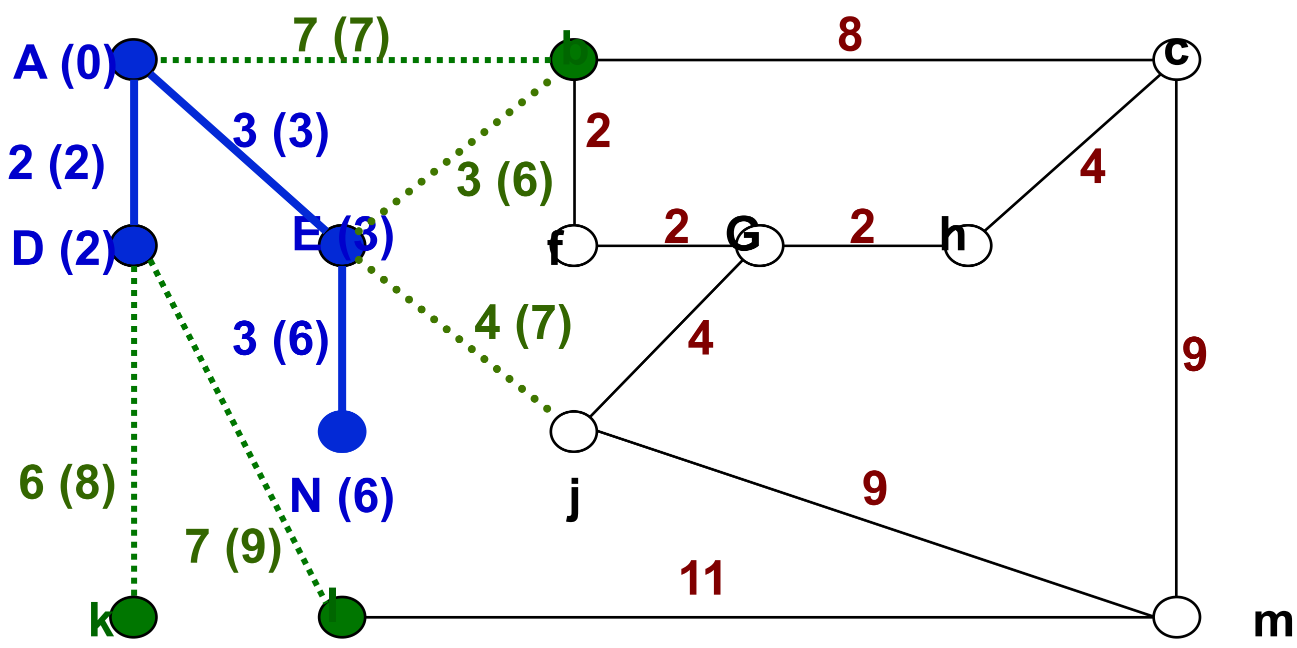
Dijkstra's Shortest path finding algorithm

- While T does not span G
  - Update frontier edges
  - For each frontier edge **e**
    - let **x** be the labelled and **y** the unlabelled endpoints of e
    - set **D(e)** = dist[x] + w(e)
  - Let **e** be a frontier edge that has the smallest D-value
  - Add edge e to T, and set **dist[y]** = D(e)



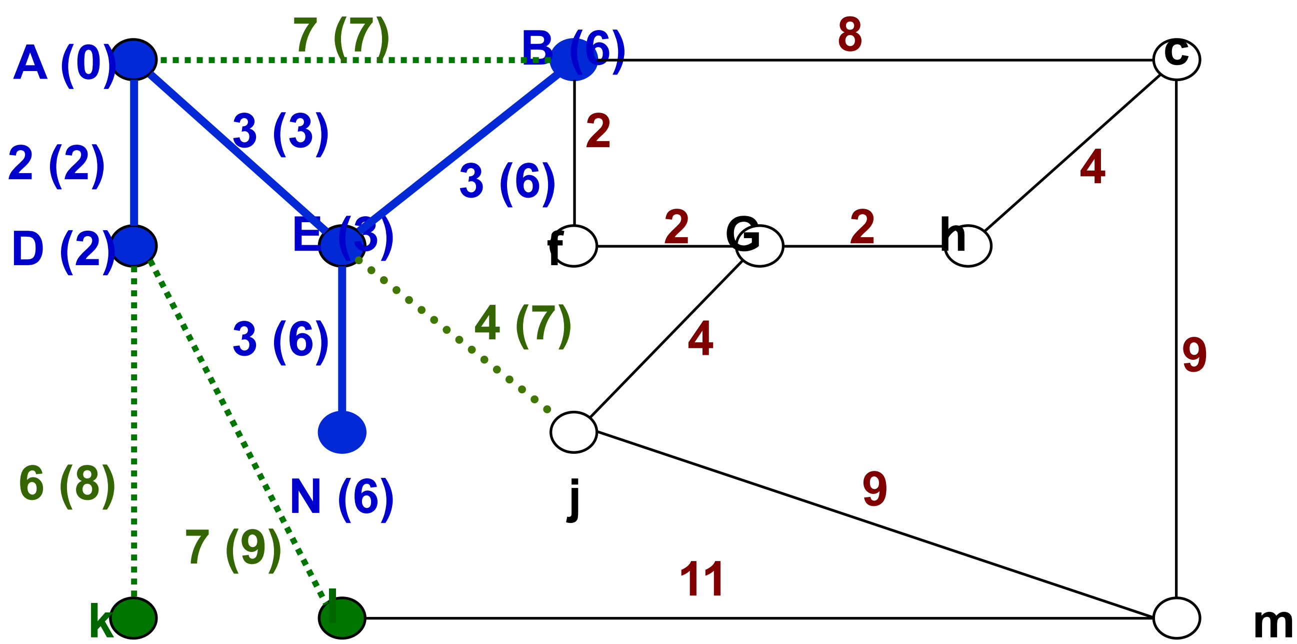
Dijkstra's Shortest path finding algorithm

- While T does not span G
  - Update frontier edges
  - For each frontier edge **e**
    - let **x** be the labelled and **y** the unlabelled endpoints of e
    - set **D(e)** = dist[x] + w(e)
  - Let **e** be a frontier edge that has the smallest D-value
  - Add edge e to T, and set **dist[y]** = D(e)



Dijkstra's Shortest path finding algorithm

- While T does not span G
  - Update frontier edges
  - For each frontier edge **e**
    - let **x** be the labelled and **y** the unlabelled endpoints of e
    - set **D(e)** = dist[x] + w(e)
  - Let **e** be a frontier edge that has the smallest D-value
  - Add edge e to T, and set **dist[y]** = D(e)



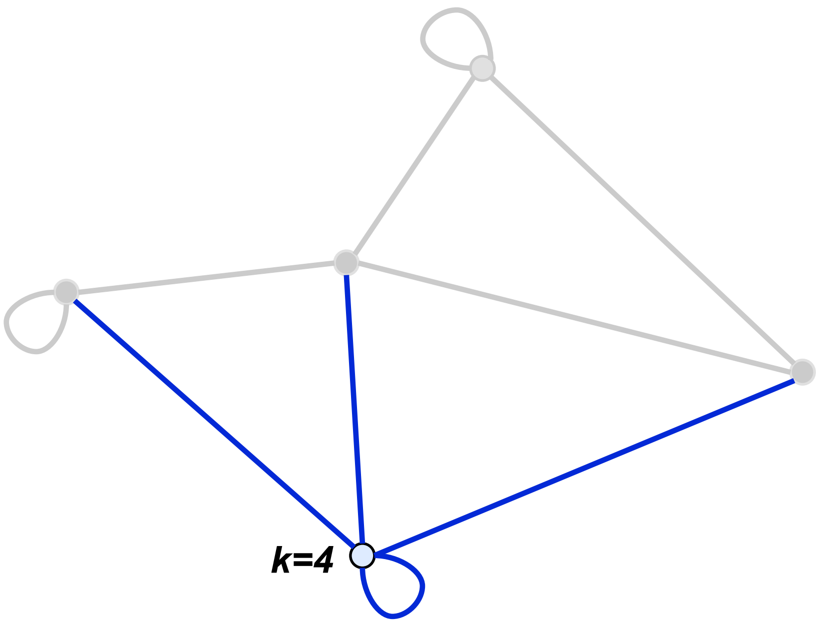
Exercise

- Follow the Dijkstra algorithm until all the vertices of the preceding graph are labelled.

Graph topology

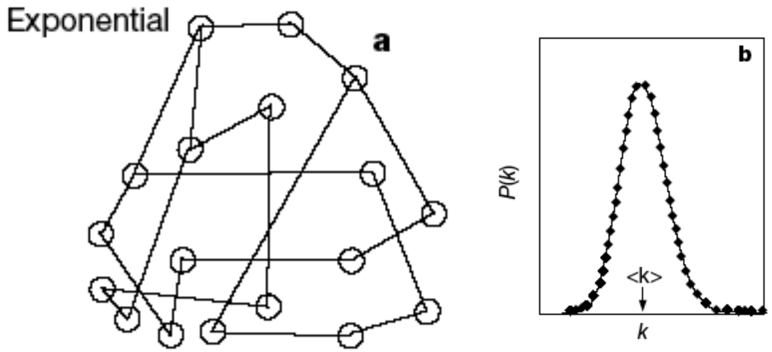
Degree - definition

- In a non-directed graph
  - The degree (k) of a node is the number of edges for which it is an endpoint.
- In a directed graph
  - The in-degree (k_{in}) of a node is the number of arcs for which it is the tail.
  - The out-degree (k_{out}) of a node is the number of arcs for which it is the head.
  - The total degree (k) of a node is the sum of in-degree and out-degree
    - $k=k_{in}+k_{out}$

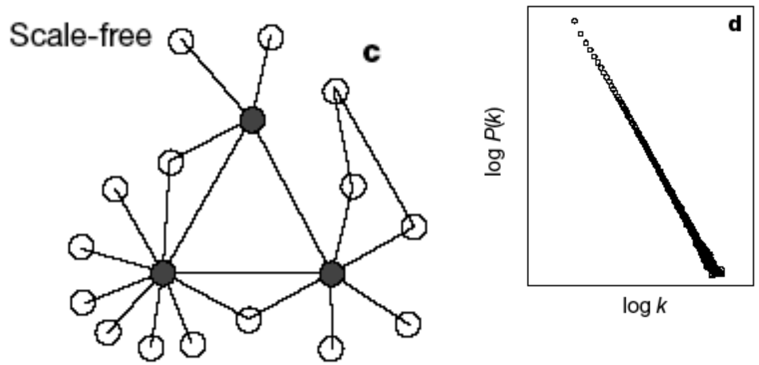


Stochastic models for the degree distribution of a graph

- Homogeneous networks
  - Erdös-Rényi model (ER model)
  - Pairs of nodes are connected with a constant random probability
  - The connectivity follows a Poisson law
    - $P(k) \sim \lambda^k e^{-\lambda} / k!$
    - λ mean number of connections per node
    - k number of connections for a given node
  - The probability of finding a highly connected node decreases exponentially with connectivity.



- Scale-free networks
  - A few nodes are highly connected, most nodes are poorly connected.
  - Can be generated randomly with a model where new nodes are preferentially connected to already established nodes
  - The connectivity follows a power law
    - $P(k) = Ck^{-\gamma} \Leftrightarrow log(P) = -y * log(k) + log(C)$
    - γ the slope of the distribution in a log-log graph.
    - k number of connections for a given node



Jeong, H., B. Tombor, R. Albert, Z.N. Oltvai, and A.L. Barabasi. 2000. The large-scale organization of metabolic networks. *Nature* 407: 651-654.

Clustering coefficient

- The clustering coefficient of a node *i* indicates the density of arcs among its neighbours.
- It is computed as the ratio between the number of arcs (*n*) between the neighbours, and the maximal number of such arcs (*M*).
- The maximal number of arcs depends on the graph type
  - Directed or undirected
  - With or without self-loops

$$C_i = \frac{n}{M}$$

Directed, self-loops

$$C_i = \frac{n}{k_i^2}$$

Directed, no self-loop

$$C_i = \frac{n}{k_i(k_i - 1)}$$

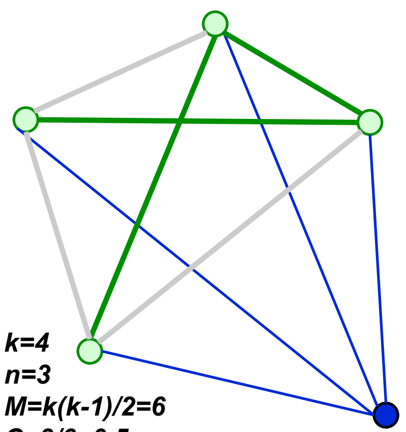
Undirected, self-loops

$$C_i = \frac{n}{M} = \frac{2n}{k_i(k_i + 1)}$$

Undirected, no self-loop

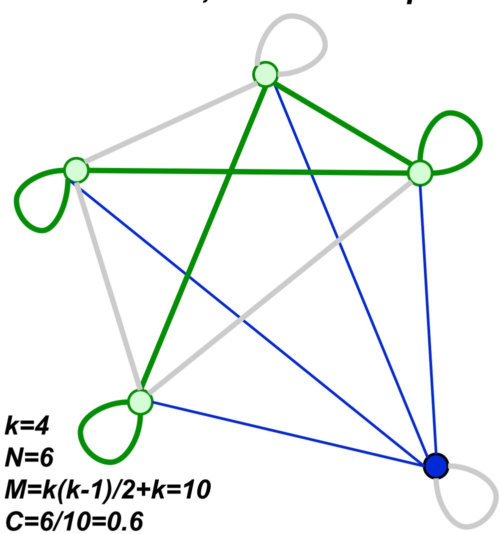
$$C_i = \frac{n}{M} = \frac{2n}{k_i(k_i - 1)}$$

Undirected, without self-loops



k=4
n=3
M=k(k-1)/2=6
C=3/6=0.5

Undirected, with self-loops



k=4
N=6
M=k(k-1)/2+k=10
C=6/10=0.6

Suggested readings

- Gross, J. & Yellen, J. (1999). Graph theory and its applications. Discrete mathematics and its applications (Rosen, K. H., Ed.), CRC press, London