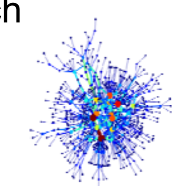
Graph Theory and Topology Design

David Tipper
Associate Professor
Graduate Telecommunications and Networking Program
University of Pittsburgh

tipper@tele.pitt.edu
Slides 4
http://www.sis.pitt.edu/~dtipper/2110.html

Top Down Network Design Approach



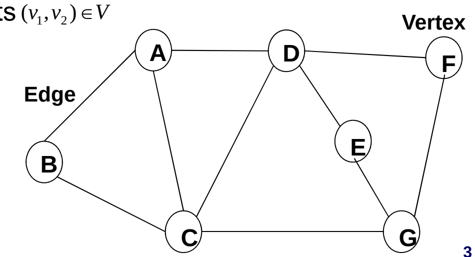
- Top down network design project approach should follow three phases:
  - Conceptual Model
    - Objectives, Requirements, Constraints
  - Logical Model
    - Technology, network graph, node location, link size, etc. (where algorithms are used to minimize cost)
  - Physical Model
    - Specific hardware/software implementations
    - (e.g., wiring diagram, repeater locations, etc.)
- Focus on Algorithms for Logical Model Design
  - Graph Theory
  - Optimization

Graphs

- Telecommunication and computer networks are naturally represented by graphs
- A graph $G = (V, E)$ is a mathematical structure consisting of two sets V and E
  - Elements of V are called *vertices* (or nodes)
    - –For example, switches, routers, crossconnects
  - Elements of E are called *edges*
    - –Communication links are edges (wired or wireless)
    - –Each edge has two endpoints $(v_1, v_2) \in V$

V ={A,B,C,D,E,F,G}

E = {(A,B),(A,C), (A,D), (B,C), …. , (F,G)}



Terminology

- Networking tends to use notation *G(N,L)* instead of *G(V, E)* for a graph where *N* is set of nodes and *L* is set of links
- A graph is *simple* if it has no loops or parallel edges.
  - *Loop*
    - Link where both endpoints are the same node. Also called a self-loop.
  - *Parallel* edges
    - A collection of two or more links having identical ends. Also called a multi-edge.
  - Focus on simple graphs
- *Degree* of a node (vertex): d_i
  - Number of links/edges out of a node (assuming same number of in and out links)
- Adjacent nodes/vertices:
  - Two nodes are adjacent if there is a link that has them as endpoints ➔ node degree d_i = number of neighbor nodes of node *i*

Terminology Cont.

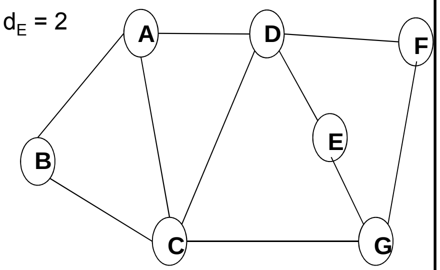
Example network: simple graph

Degree of Node A $d_A = 3$, Degree of Node E $d_E = 2$

A and B are adjacent, A and E not

Size of graph characterized by number of nodes |N| and number of links |L|

Example network: |N| = 7, |L| = 10



- Can represent graph by *Adjacency matrix A* which is $|N| \times |N|$ matrix where

  $a_{ij} = 1$ if link exist between nodes *i* and *j*

  $a_{ij} = 0$ otherwise

A =

	A	B	C	D	E	F	G
A	-	1	1	1	0	0	0
B	0	-	1	0	0	0	0
C	1	1	-	1	0	0	1
D	1	0	1	-	1	1	0
E	0	0	0	1	-	0	1
F	0	0	0	1	0	-	1
G	0	0	1	0	1	1	-

Paths and Cycles

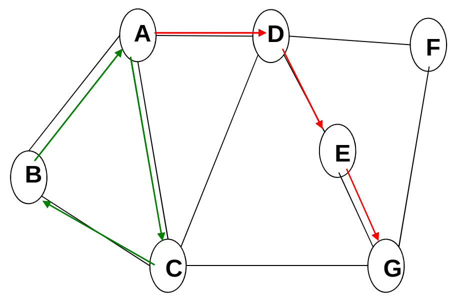
- Path from node A to node Z:

  An alternating sequence of nodes and links, representing a continuous traversal from vertex A to vertex Z.
- Trail: a path with no repeated edges.
- Cycle: a path starting and ending on the same node
- Connected graph:

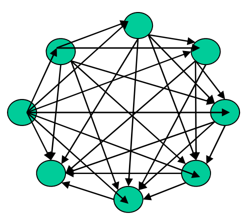
  A graph in which every pair of distinct nodes has a path between them.
- Weighted Graph:
  - A graph *G(N,L)* is weighted if there is a value w_{ij} associated with each link $l_{ij} \varepsilon L$
    - For example, link speed, cost, etc.
  - We often denote this graph *(G, W)* or *G(N,L,W)*.

Terminology Cont.

Example: Path from A to G is given by (A,D),(D,E),(E,G)
Cycle at A is given by (A,C), (C,B), (B,A)
Example is a connected Graph

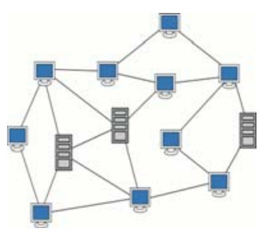


Graph Types



Complete Graph: every node is connected to every other node – also called a *Full Mesh*

N node network – every node has degree *(N-1)*

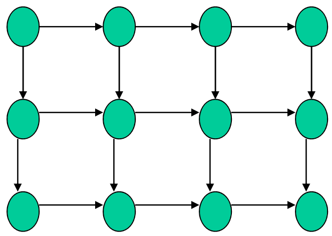


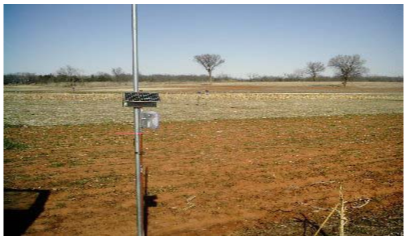
- *Mesh Graph*
  - Each node having degree *2 or more* and forming a connect graph in which every pair of distinct nodes has a path between them.

Graph Types

Grid Graph: Nodes have a regular grid pattern:

Occurs in parallel computing, sensor networks , etc.

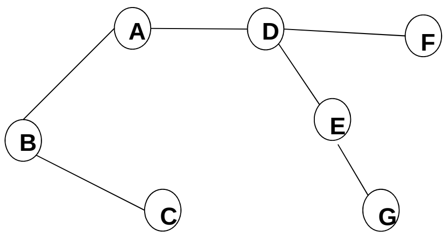




Telcom 2110 9

Graph Types

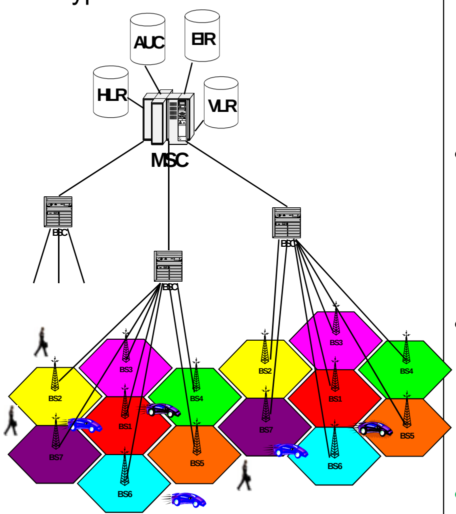
- Tree: a connected, simple graph without cycles.
- Any tree with *N* nodes has *N-1* links
- Trees often used in access networks



Telcom 2825 10

Tree Terminology

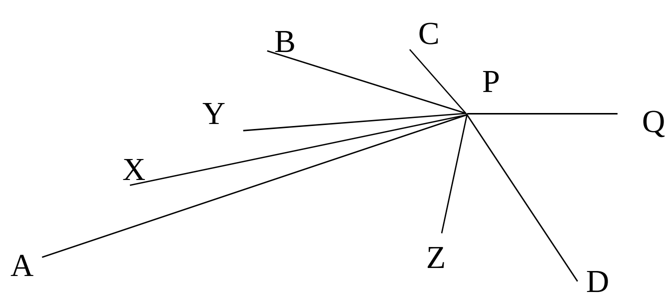
Typical Cellular Network



- *Root*: One node of a tree may be designated as a root (has no parent only children)
- Each node (besides root) has a single *parent* node which is the node closest to the root
- Each node has zero or more *child* nodes which are the adjacent nodes farthest from the root
- *Leaf*: a node without a child

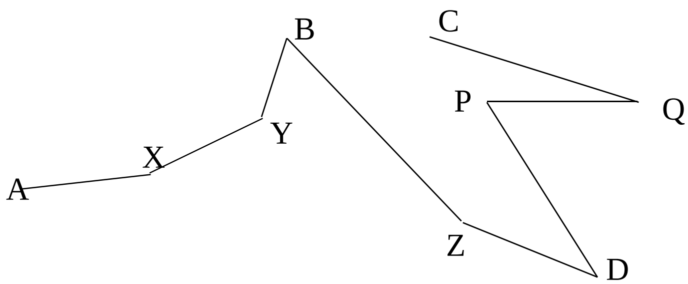
Graph Types

- A tree is a *STAR* if only 1 node has degree >1



Graph Types

- A ***CHAIN*** is a tree with no nodes of degree >2

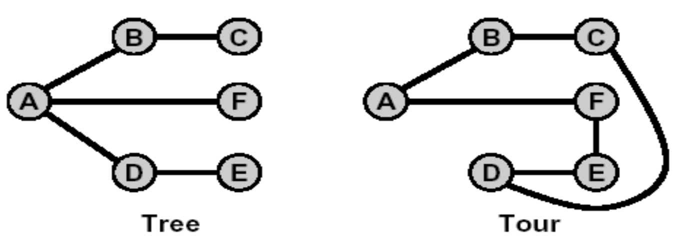


•Trees are usually the cheapest network design

–However have poor reliability

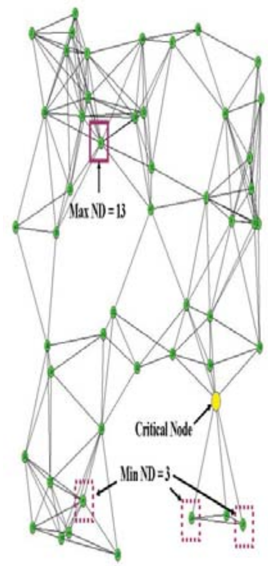
Graph Types

- In graph theory, a *tour* refers to a possible solution of the traveling salesman problem (TSP). Given a set of Nodes $N = \{n_1, n_2, \ldots n_N\}$ a *tour* is a set of N links $l \in L$ such that each node N has degree 2 and the graph is connected – in networking this is a *ring* topology
- *Rings* are used when reliability is important



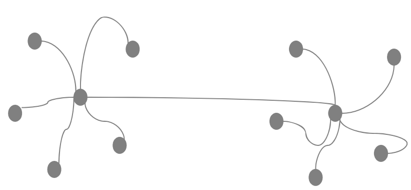
Graph Analysis

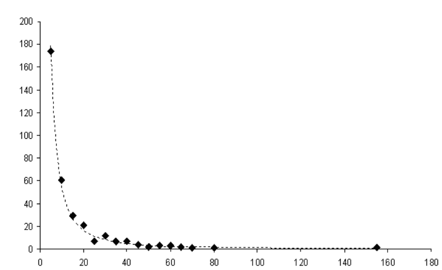
- Basic graph theory analysis to study/compare network topologies
- Some Typical Metrics
  - Maximum Node degree
  - Average node degree
  - Minimum node degree
  - Average path length between a node pair
  - Average shortest path length network wide
  - Network Diameter
    - length of longest shortest path in the network
  - Number of critical points in graph
    - Link/node whose loss partitions graph
  - K –connectivity
    - G is k – connected in removal of any combination of k-1 nodes doesn't partition the graph
  - Etc..



Small World Graphs/Networks

- A property of some networks is ``small world" or scale free behavior
  - Small number of hops to reach most people
  - Clustering into Neighborhoods
- Used to model social networks





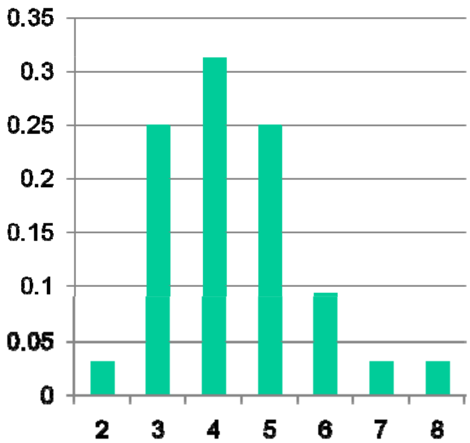
- Scale-Free Networks

Distribution of node degree has a power law behavior $\sim k^{-r}$ where k = # links; r > 1, typically 2 < r < 3

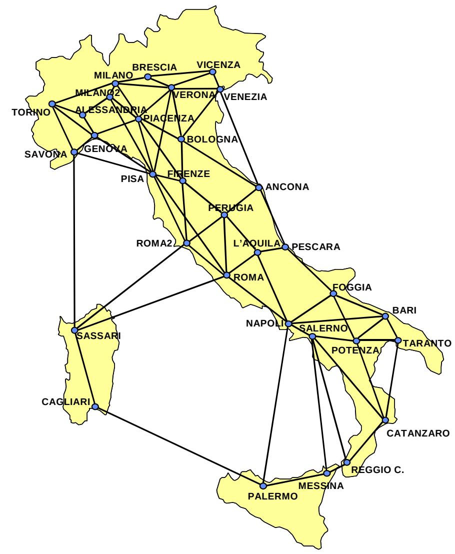
Simple test for scale free is to plot a histogram of node degree – test power law behavior

Telecom Italia Backbone

Testing if Scale Free
get frequency histogram

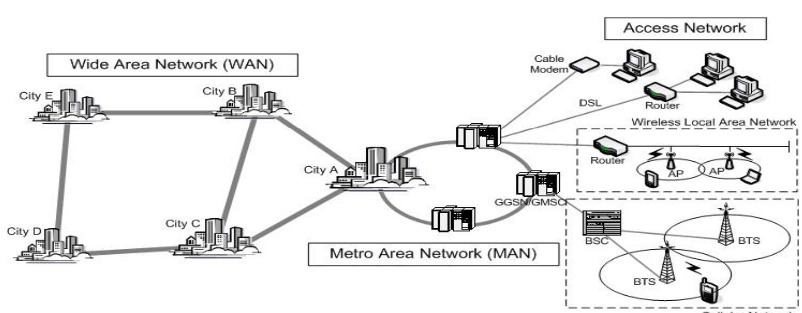


Not scale free!



Network Topologies

- Most networks a mix of trees, rings, mesh – depending on network type, cost/traffic/reliability
- Need to know how to determine good topologies for
  - Tree, Ring and Mesh
  - Use graph theory derived algorithms for Tree and Rings



Design of Trees

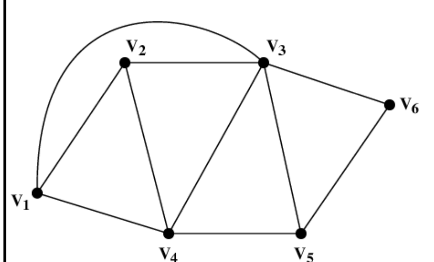
- Many algorithms for design and types of trees
  - Minimum Spanning Trees, Shortest Path Trees, etc.
- Spanning Trees and Subgraphs
  - Subgraph of graph *G* obtained by selecting number of links and nodes from *G*
    - For each link, the two nodes incident on that link must be selected
  - Give graph $G(N,L)$, graph $G'(N',L')$ is a subgraph of G iff

    $N' \subseteq N$ and $L' \subseteq L$ and

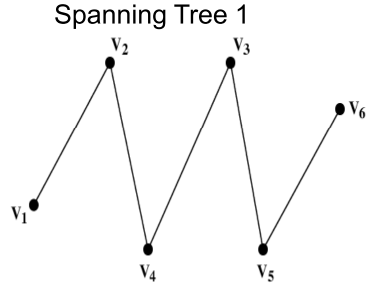
    $\exists\ l' \in L'$, if l' incident on e' and w' then $e', w' \in N'$
  - A *spanning* subgraph includes *all the nodes* of *G*
  - A tree *T* is a *spanning tree* of *G* if *T* is a spanning subgraph of *G*
    - Not usually unique➔ typically many spanning trees

Spanning Tree Examples

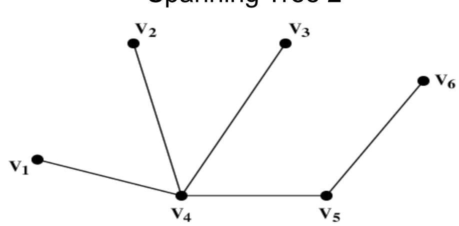
Network Graph Considered



Spanning Tree 1

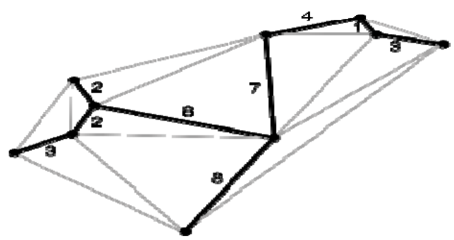


Spanning Tree 2



Finding the MST

- The *Minimal Spanning Tree (MST)*
  - A spanning tree of *G* whose total weight is a minimum ➔ minimum cost spanning tree
  - Can have many MSTs – all with same cost
- MSTs are used in for network designs when have just few nodes and cost is dominant factor (Access networks)
- Two algorithms *Kruskal* and *Prim*

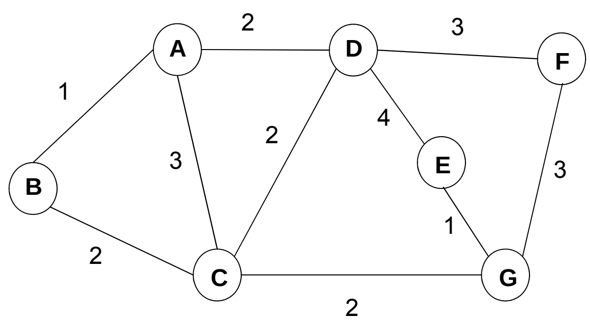


Prim's Algorithm

- Algorithm
  - given a weighted graph *G(N,L,W)* starts by selecting a node
  - adding the "least expensive link"
  - iterates until tree is built
- U = set of nodes in MST
- V' = set of nodes that are NOT in MST but are adjacent to nodes in U
  1. Place any node in U and update V'
  2. Find the link with smallest weight that connects a node in V' to a node in U
  3. Add that edge to the tree and update U & V'.
  4. Repeat 2 & 3 until all nodes are included | U | = | N |

Algorithm Example

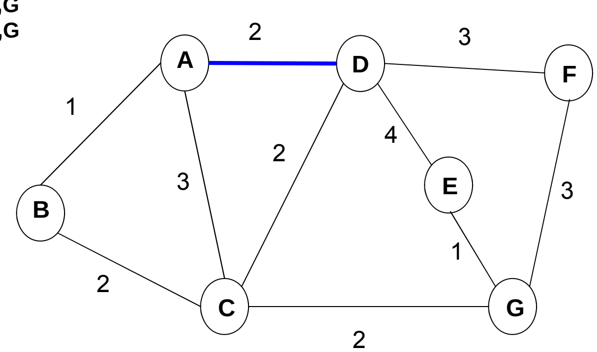
Apply Prim algorithm to the graph below



Prim's Algorithm Example

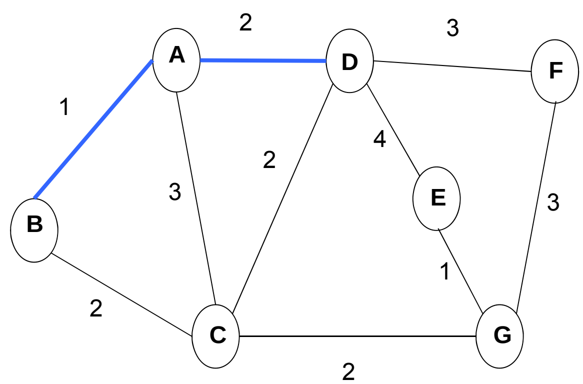
Arbitrarily pick node D to start with – min cost link to a node in V' is (D,A)

Iteration	U	V'
0	D	A,B,C,E,F,G
1	D,A	B,C,E,F,G



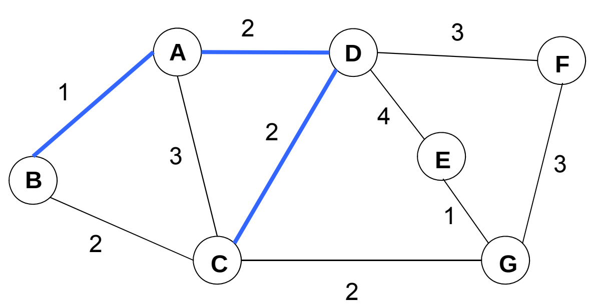
Prim's Algorithm Example

Iteration	U	V'
0	D	A,B,C,E,F,G
1	D,A	B,C,E,F,G
2	D,A,B	C,E,F,G



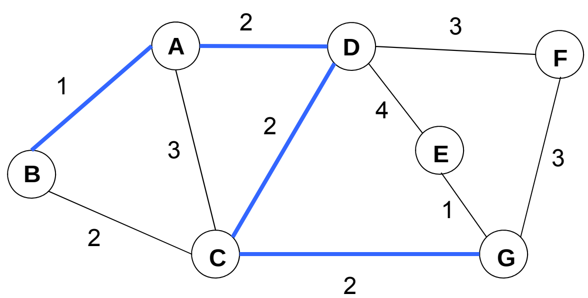
Prim's Algorithm Example

Iteration	U	V	
0	D	A,B,C,E,F,G	
1	D,A	B,C,E,F,G	
2	D,A,B	C,E,F,G	
3	D,A,B,C	E,F,G	*<= arbitrarily pick (D,C) link rather than (B,C)*



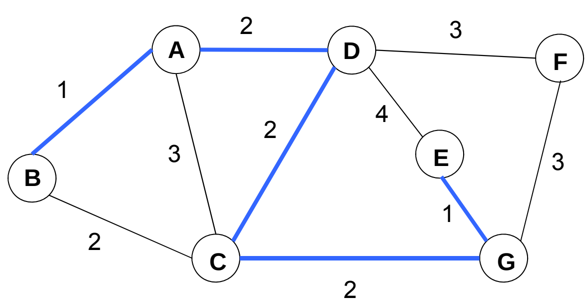
Prim's Algorithm Example

Iteration	U	V
0	D	A,B,C,E,F,G
1	D,A	B,C,E,F,G
2	D,A,B	C,E,F,G
3	D,A,B,C	E,F,G
4	D,A,B,C,G	E,F



Prim's Algorithm Example

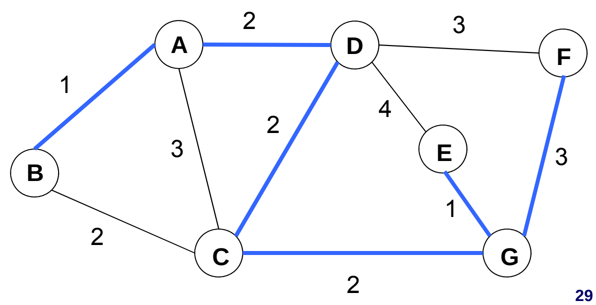
Iteration	U	V
0	D	A,B,C,E,F,G
1	D,A	B,C,E,F,G
2	D,A,B	C,E,F,G
3	D,A,B,C	E,F,G
4	D,A,B,C,G	E,F
5	D,A,B,C,G,E	F



Prim's Algorithm Example

Iteration	U	V'
0	D	A,B,C,E,F,G
1	D,A	B,C,E,F,G
2	D,A,B	C,E,F,G
3	D,A,B,C	E,F,G
4	D,A,B,C,G	E,F
5	D,A,B,C,G,E	F
6	D,A,B,C,G,E,F	*<= arbitrarily pick (G,F) link rather than (D,F) link*

MST is complete weight is 11



Kruskal's Algorithm

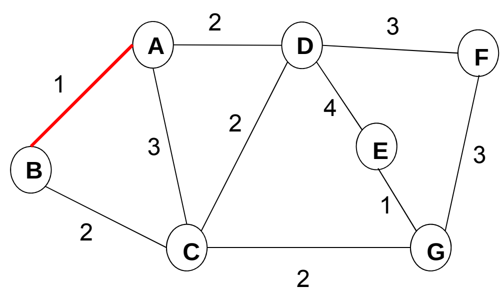
- Kruskal achieves the MST by starting with a graph and picking out edges based on cost
- 1. Check that the graph G is connected. If it is not connected stop
- 2. Sort the edges of the graph G in *ascending* order of weight.
- 3. Mark each node as a separate component.
- 4. Examine each of the sorted edges:

  if the edge connects two separate components, add it ; otherwise, discard and go to step1

Algorithm Example

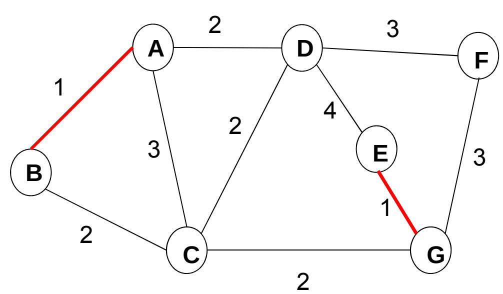
Apply Kruskal's algorithm to the graph below

Pick one of the edges with minimum weight
Arbitrarily pick (A,B) rather than (E,G)



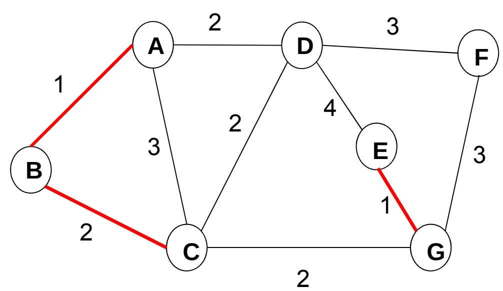
Algorithm Example

Iteration 2 pick (E,G) as it has minimum weight



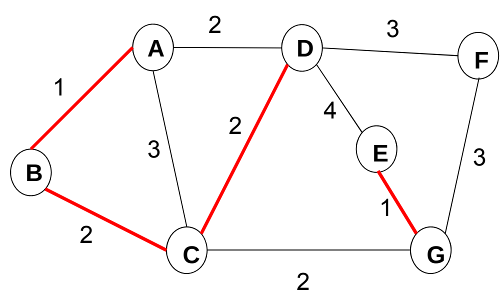
Algorithm Example

Iteration 3
Arbitrarily pick (B,C) out of possible choices (B,C), (A,D), (C,D),(C,G)



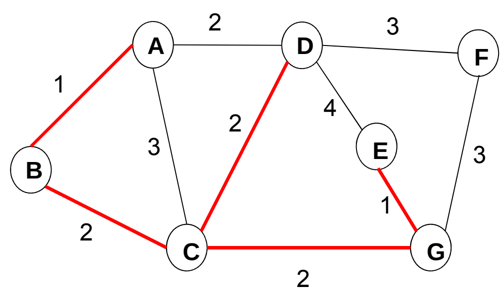
Algorithm Example

Iteration 4
Arbitrarily pick (C,D) out of possible choices (A,D), (C,D),(C,G)



Algorithm Example

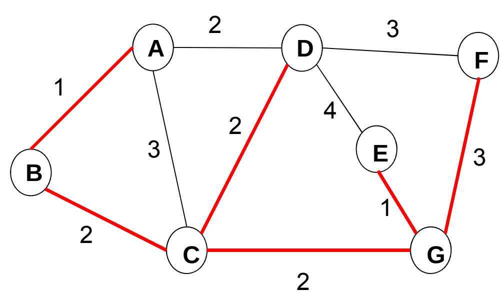
Iteration 5 pick (C,G) as (A,D) is not a valid choice (A and D are in same component)



Algorithm Example

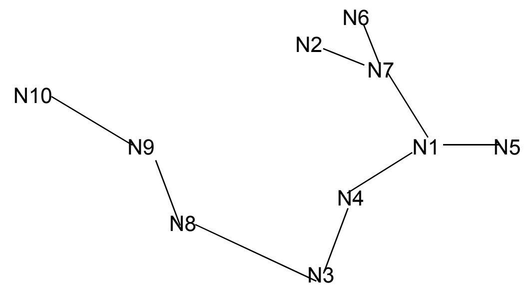
Iteration 6 pick (G,F) from possible choices (D,F), (G,F)

MST is complete weight is 11



MST's Drawbacks

- An MST for 10 nodes



MSTs don't scale well when traffic is internal – note graph above is beginning to have a leggy look, which means that some traffic is taking a circuitous route between its source and destination.

Shortest-Path Trees (SPT)

- Shortest Path

  Given a weighted graph *(G,W)* and nodes n_1 and n_2, the shortest path from n_1 to n_2 is a path P such that the sum of link weights along the path $\sum_{e \in P} w(e)$ is a minimum.

- Shortest Path Tree
  - Given a weighted graph (G,W) and a node n_1, a shortest – path tree rooted at n_1 is a tree *T* such that, for any other node $n_2 \in$ G, the path from n_1 to n_2 in the tree *T* is a shortest path between the nodes.

- SPT vs. MST
  - SPT ***cost more***, but will have lower link utilization and lower delay, smaller average hop count

Finding a Shortest Path Tree

- Given a connected graph G and a node selected to be a root
- Dijkstra's algorithm can be used to find a shortest path tree
- The algorithm is similar to Prim's in that one iteratively builds a tree
  - Let *N* = set of Nodes
  - *S* = source node
  - *U* = set of nodes incorporated so far
  - *W()* is the link cost, specifically *w(i,j)* is the cost from node i to node *j* , *w(i,j)* = ∞ if the two vertices are not directly connected
  - *d_min* is the currently known minimum cost path from node *s* to node *k*

Finding a Shortest Path Tree

- Dijkstra's Algorithm
- 1. Initialization: Mark every node as unscanned and *U = {s}, d_min(k) = w(s,k)* for $k \neq s$
- 2. Loop until you have scanned all the nodes.

  A. Find the node *x* not in tree *T* with the minimum cost path from *s,* add *x* to *T*

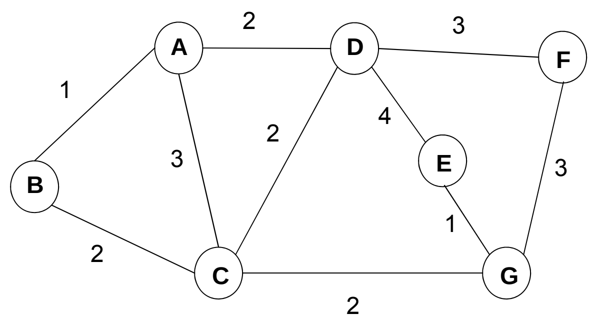
  B. Update the minimum cost paths

  $d\_min(k) = min\{d\_min(k), d\_min(x) + w(x,k)\}$
- Terminate when all nodes added to T
- Requires |N| iterations

Algorithm Example

Apply Dijkstra's algorithm to find a SPT rooted at D

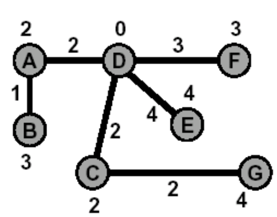
Iteration	T	d_min(A)	Path	d_min(B)	Path	d_min(C)	Path	d_min(E)	Path	d_min(F)	Path	d_min(G)	Path
1	{D}	2	(D,A)	∞	-	2	(D,C)	4	(D,E)	3	(D,F)	∞	-
2	{D,C}	2	(D,A)	4	(B,C),(C,D)	2	(D,C)	4	(D,E)	3	(D,F)	4	(G,C),(C,D)



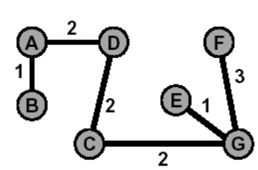
Algorithm Example

Iteration	T	d_min(A)	Path	d_min(B)	Path	d_min(C)	Path	d_min(E)	Path	d_min(F)	Path	d_min(G)	Path
1	{D}	2	(D,A)	∞	-	2	(D,C)	4	(D,E)	3	(D,F)	∞	-
2	{D,C}	2	(D,A)	4	(B,C),(C,D)	2	(D,C)	4	(D,E)	3	(D,F)	4	(G,C),(C,D)
3	{D,C,A}	2	(D,A)	3	(B,A),(A,D)	2	(D,C)	4	(D,E)	3	(D,F)	4	(G,C),(C,D)
4	{D,C,A,F}	2	(D,A)	3	(B,A),(A,D)	2	(D,C)	4	(D,E)	3	(D,F)	4	(G,C),(C,D)
5	{D,C,A,F,B}	2	(D,A)	3	(B,A),(A,D)	2	(D,C)	4	(D,E)	3	(D,F)	4	(G,C),(C,D)
6	{D,C,A,F,B,E}	2	(D,A)	3	(B,A),(A,D)	2	(D,C)	4	(D,E)	3	(D,F)	4	(G,C),(C,D)
7	{D,C,A,F,B,E,G}	2	(D,A)	3	(B,A),(A,D)	2	(D,C)	4	(D,E)	3	(D,F)	4	(G,C),(C,D)

SPT is a Star topology



MST



Prim – Dijkstra Trees

- MSTs have high delay – but are cheap
- SPTs have lower delay and utilization but more expensive
- Prim-Dijkstra algorithm – interpolates between MST and SPT (comprise)
- Algorithms :

1) Prim's: $\min_{neighbors} dist(node, neighbor)$

2) Dijkstra's:

$$\min_{neighbors}(dist(root, neighbor) + dist(neighbor, node))$$

3) Prim-Dijkstra's: $0 \le \alpha \le 1$

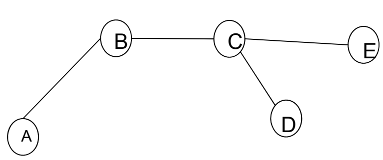
$$\min_{neighbors}(\alpha \times dist(root, neighbor) + dist(neighbor, node))$$

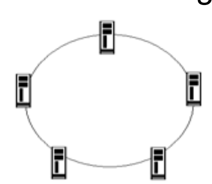
Rings

- A tree maybe too unreliable to be a good network design as they are subject to single point of failure
- Consider the reliability of Tree vs. Ring

  Let p = probability of a link failure
- Five Node Tree

Five Node Ring





P(No Failure) = $(1-p)^4$
P(Failure) = $1 – (1-p)^4 = 1 – (1 – 4p + 6p^2 – 4p^3 + p^4)$
$= 4p - 6p^2 + 4p^3 - p^4$

P(Failure) = $1 - (1-p)^5 – 5p(1-p)^4$
P (Failure) = $10p^2(1-p)^3 + 10p^3(1-p)^2 + 5p^4(1-p) + p^5$

Rings and Reliability

- Comparing the reliability of Trees vs Rings

p	Tree	Ring
.1	.3439	.0815
.01	.0394	9.8×10^{-4}
.001	.004	9.98×10^{-6}
.0001	3.9994×10^{-4}	9.998×10^{-8}
.00001	3.9994×10^{-5}	9.9998×10^{-10}
.000001	4×10^{-6}	1×10^{-11}

- How can one find a good ring topology?

Traveling Salesman Problem (TSP)

- Number of tours is in a set of N nodes is $(N-1)!/2$
- Finding a tour/ring is equivalent to the Traveling Salesman Problem (TSP)
- Given a set of nodes $(n_1, n_2, \ldots, n_N)$
  and a distance/cost function $d : N \times N \rightarrow \Re^+$,
  the traveling salesman problem is to find the tour such that
  $\sum_{i=1}^{N} d(n_i, n_{i+1})$ is a minimum.
- TSP is a tough problem (NP Hard)
- Solve using use heuristic algorithms.

Nearest-neighbor Algorithm

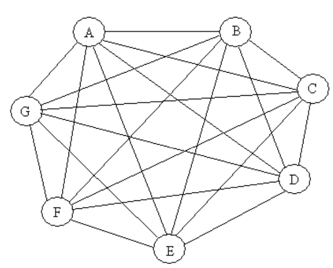
1. Start at a node we call *root* and set *current_node = root.*
2. Loop until we have all the nodes in the tour.
   - Find the node closest (i.e., min cost or distance) to the *current_node* that is not in the tour. We call this *best_node.*
   - Create an edge between *current_node* and *best_node*.
   - Reset the *current_node* to the *best_node*.
3. Finally create an edge between the last node and the root to complete the tour.

Nearest Neighbor Example

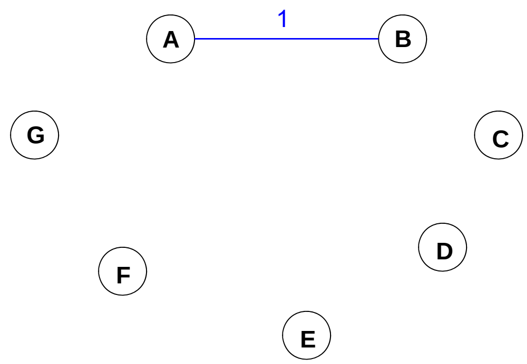
- Example: Start at node A

Table 6.1 Example Network Link Costs

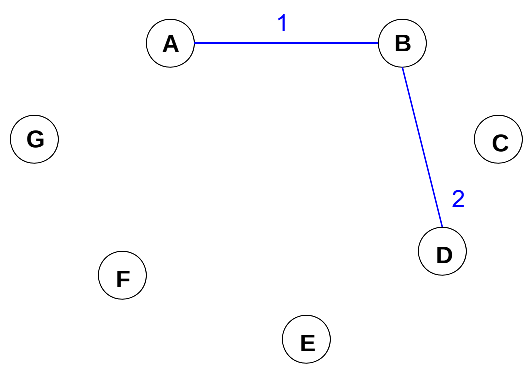
Node	B	C	D	E	F	G
A	5	6	9	10	11	15
B		9	6	6	8	17
C			7	9	8	12
D				10	5	11
E					14	9
F						8



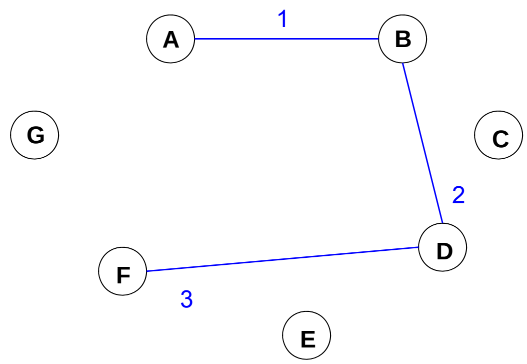
Nearest Neighbor Example



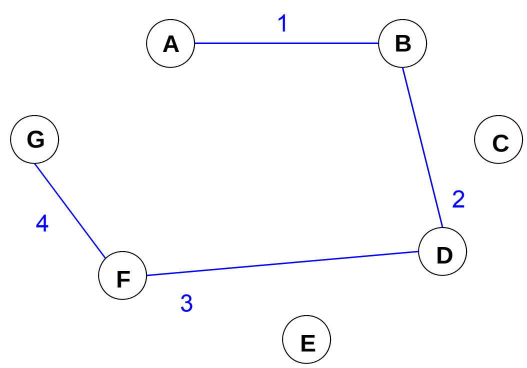
Nearest Neighbor Example



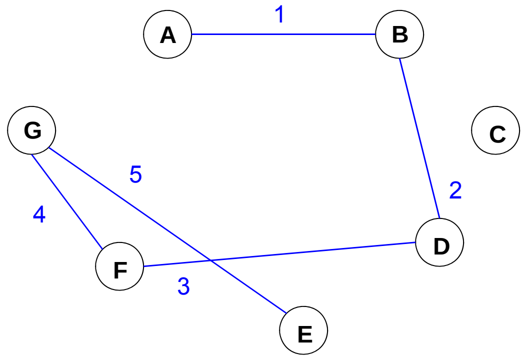
Nearest Neighbor Example



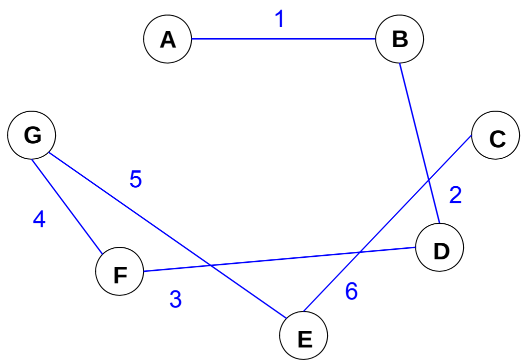
Nearest Neighbor Example



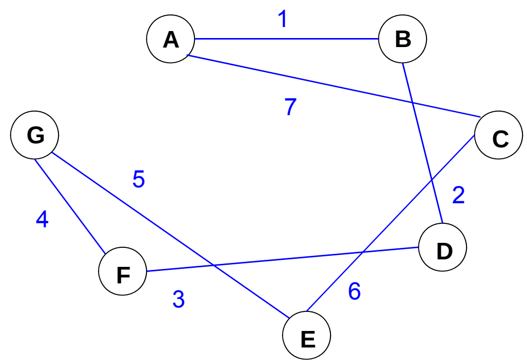
Nearest Neighbor Example



Nearest Neighbor Example



Nearest Neighbor Example



Total Cost = 50

Nearest-neighbor Algorithm

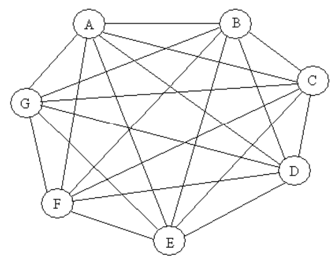
- Observation:
  - Good (?):
    We are trying to produce a short tour, we will always move to the best possible next location.
  - Bad (?):
    When we look at the figure produced, we can see the lines may cross frequently.
- Several improved version of nearest-neighbor in the literature - will look at optimization based approaches later
- Simple improvement is grow ring/tour from both ends
  - That is when finding best node to move to look at option from both ends of current partial tour

Nearest Neighbor

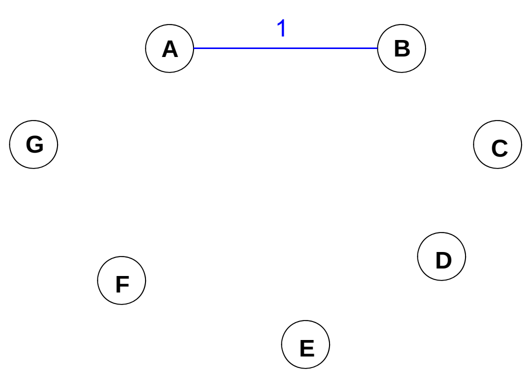
- Example: Start at node A

Table 6.1 Example Network Link Costs

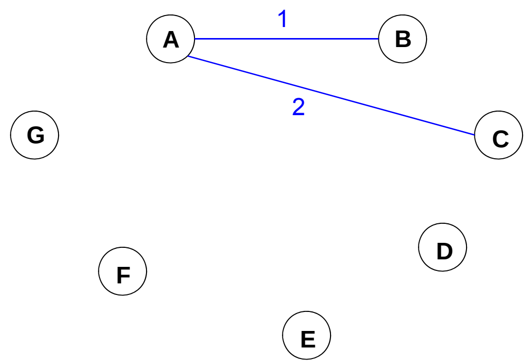
Node	B	C	D	E	F	G
A	5	6	9	10	11	15
B		9	8	8	8	17
C			7	9	7	12
D				10	5	11
E					14	9
F						8



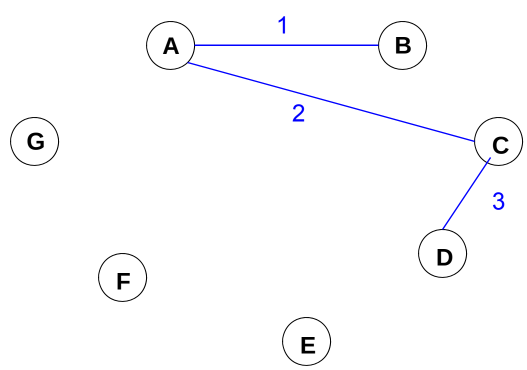
Nearest Neighbor Example



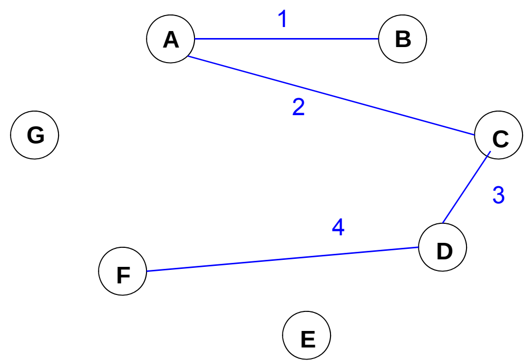
Nearest Neighbor Example



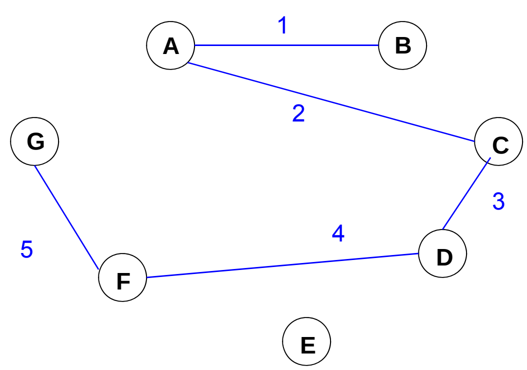
Nearest Neighbor Example



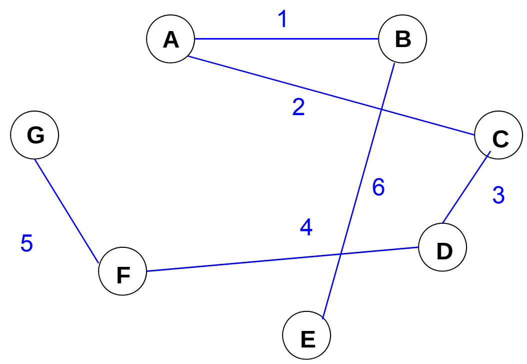
Nearest Neighbor Example



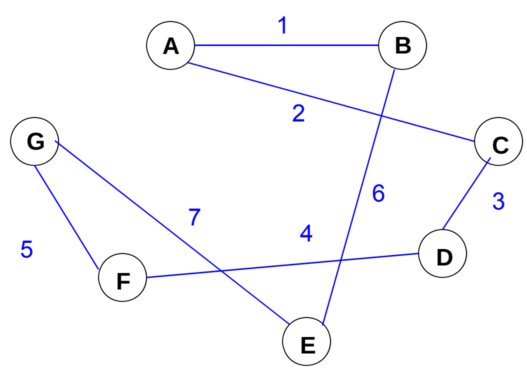
Nearest Neighbor Example



Nearest Neighbor Example



Nearest Neighbor Example



Total Cost = 48

(Rings) Do Not Scale

Given uniform traffic any Ring of N nodes has $\overline{hops} = \frac{N+1}{4}$ if n is odd and $\frac{N^2}{4(N-1)}$ if n is even.

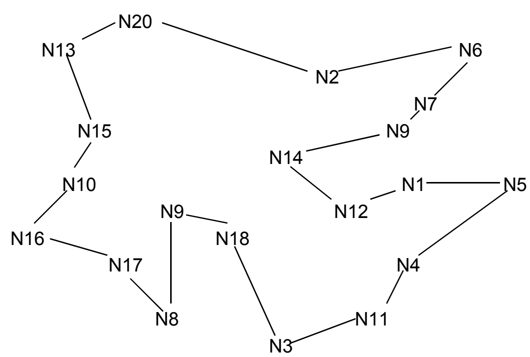
- Comparison of average number of hops for MST and TSP:

Number of nodes	$\overline{hops}_{MST}$	$\overline{hops}_{TSP}$
5	1.8	1.5
10	3.1778	2.777
20	4.4158	5.263
50	8.5159	12.755
100	13.9479	25.252

Improving Ring Topologies

- Can reduce hop count by adopting a multi-ring topology.
- Topology is a set of interconnected rings

- Example, a TSP tour on 20 nodes. The average number of hops is 5.263. We want to reduce the average hop count but keep the 2-connectivity.



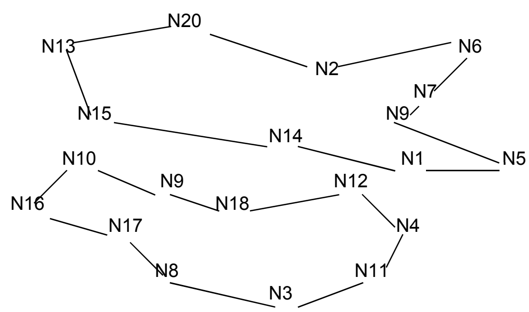
Divide and Conquer

- Use a Divide and Conquer approach
- Divide nodes into disjoint subset , construct ring for each subset, then join rings
- Example
  - Divide the 20 nodes into 2 “compact” clusters of 10 nodes each. Call these clusters C1 and C2.

    (We might divide the 20 nodes by ranges of their coordinates, for example, to create the 2 clusters.)
  - Use the nearest-neighbor algorithm to design 2 TSP tours on each cluster.
  - Select v1 $\in$C1 and v2 $\in$C2 to be the 2 nodes such that the distance is the minimum.
  - Now select v3 $\in$C1-v1 and v4 $\in$C2-v2 to be the 2 nodes such that the distance is the minimum.
  - Add the edges (v1,v2), (v3,v4) to the design.

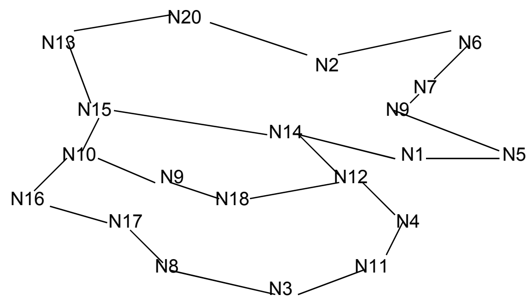
Divide and Conquer

- Grouping into 2 groups of 10 nodes. Then running the nearest neighbor algorithm gives two rings as below. Note that the average hop count is reduced



Divide and Conquer

- Grouping into 2 groups of 10 nodes. Then running the nearest neighbor algorithm gives two rings as below. Joining the two rings at their closet points results in

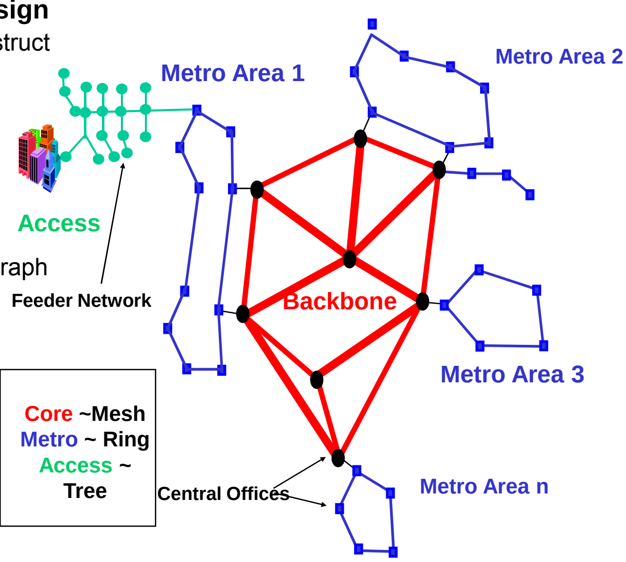


Level 3 N. American Network

Snapshoot of their backbone in mid 90's

Typical Network Design

- **Network Topology Design**
- Need to know how to construct
  - •Trees
  - •Rings
  - •Mesh networks



- Algorithms adopted from graph theory are used for Trees and Rings

Summary

- Basic Graph theory terminology and techniques
  - Analysis useful to compare/evaluate designs
- Trees and Rings are often used in access networks
- Trees
  - MST (Prim, Kruskal algorithrms)
  - SPT
  - Prim-Dikjistra Trees
- Rings
  - Better reliability than trees
  - Nearest neighbor, Improved nearest neighbor
  - Multi-Ring