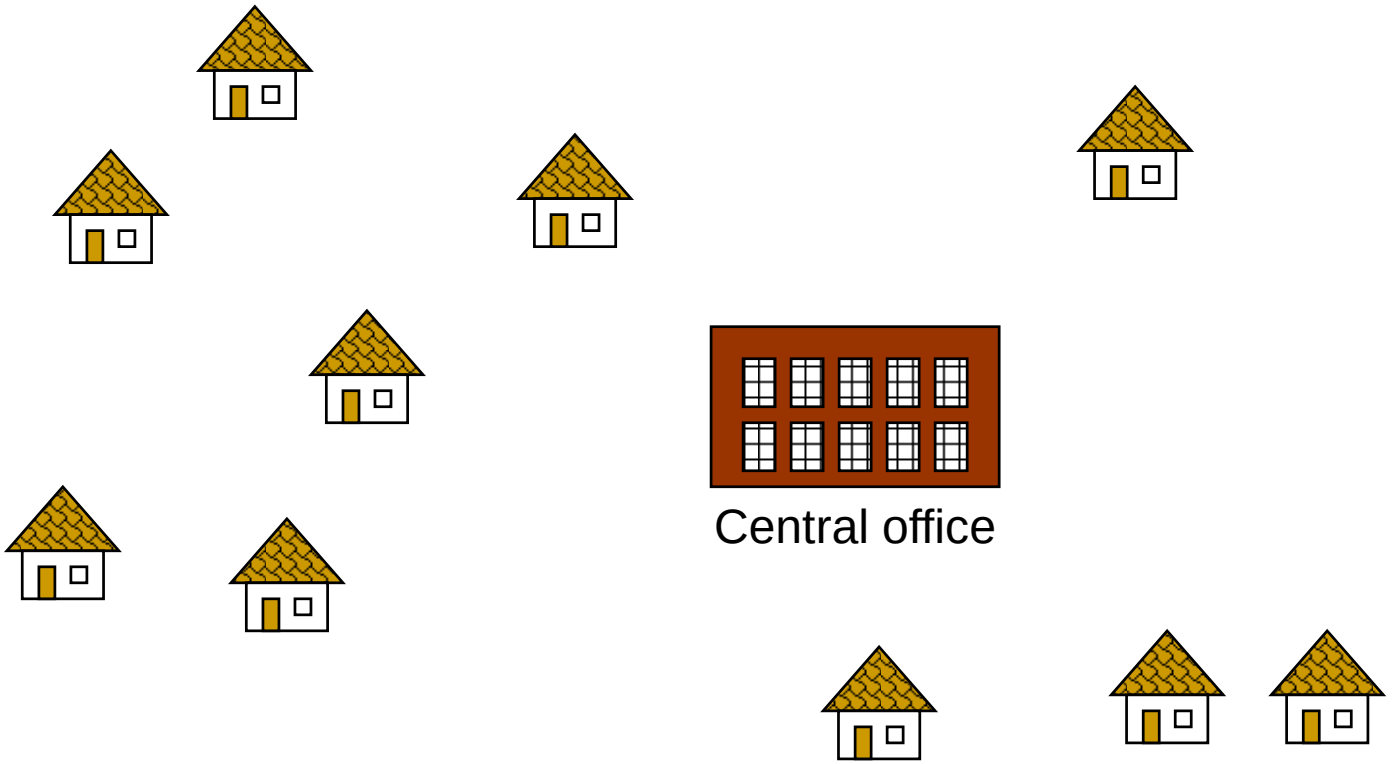
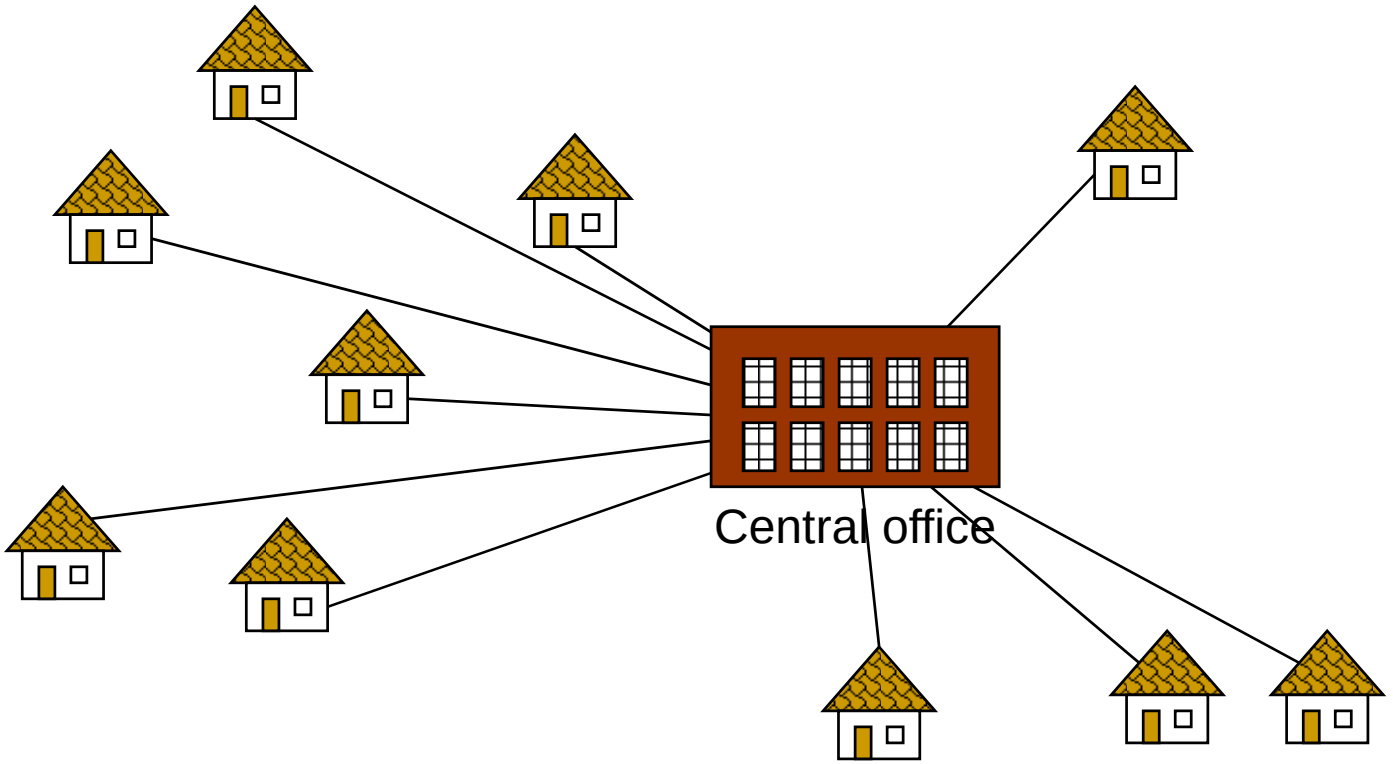
Minimum SPANNING TREE
Greedy Algorithm

Bibhudatta Sahoo,
National Institute of Technology Rourkela

Problem: Laying Telephone Wire

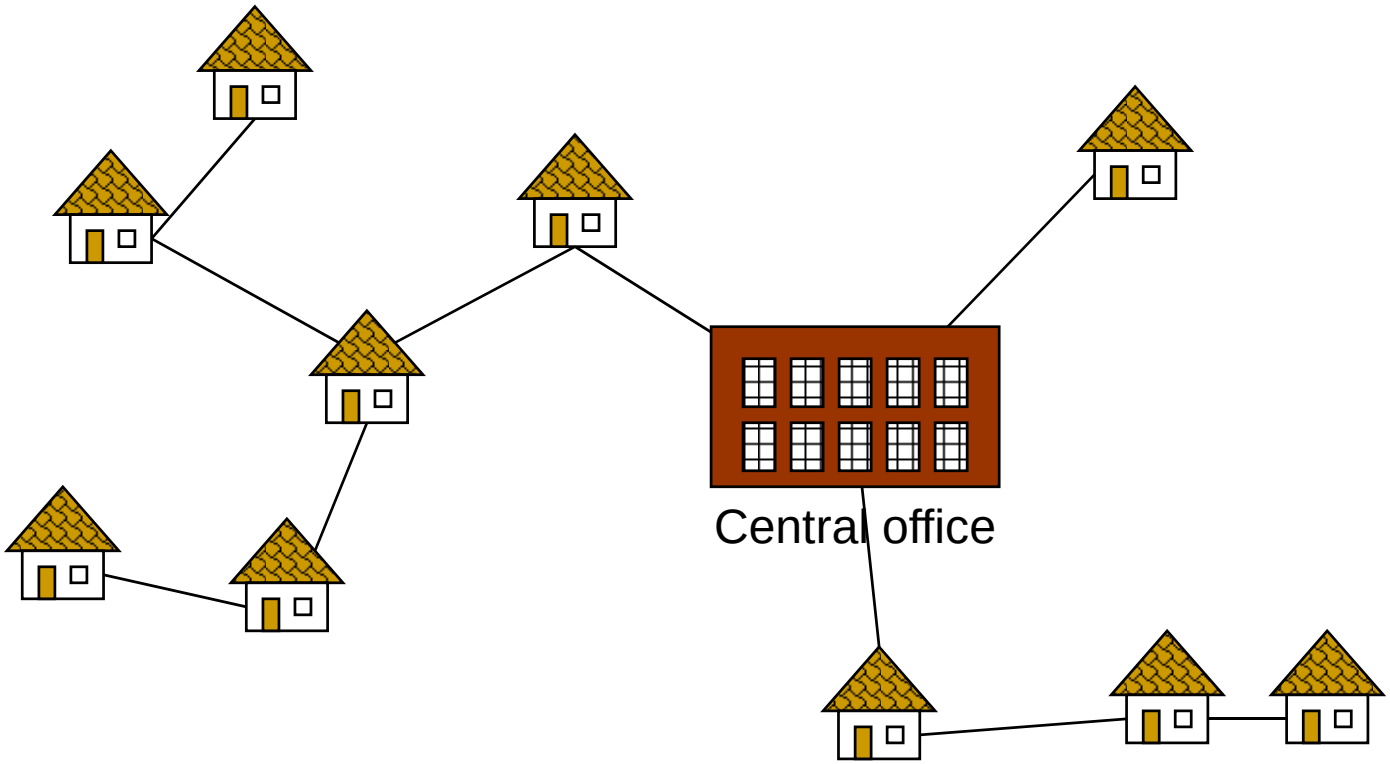


Wiring: Naïve Approach



Expensive!

Wiring: Better Approach

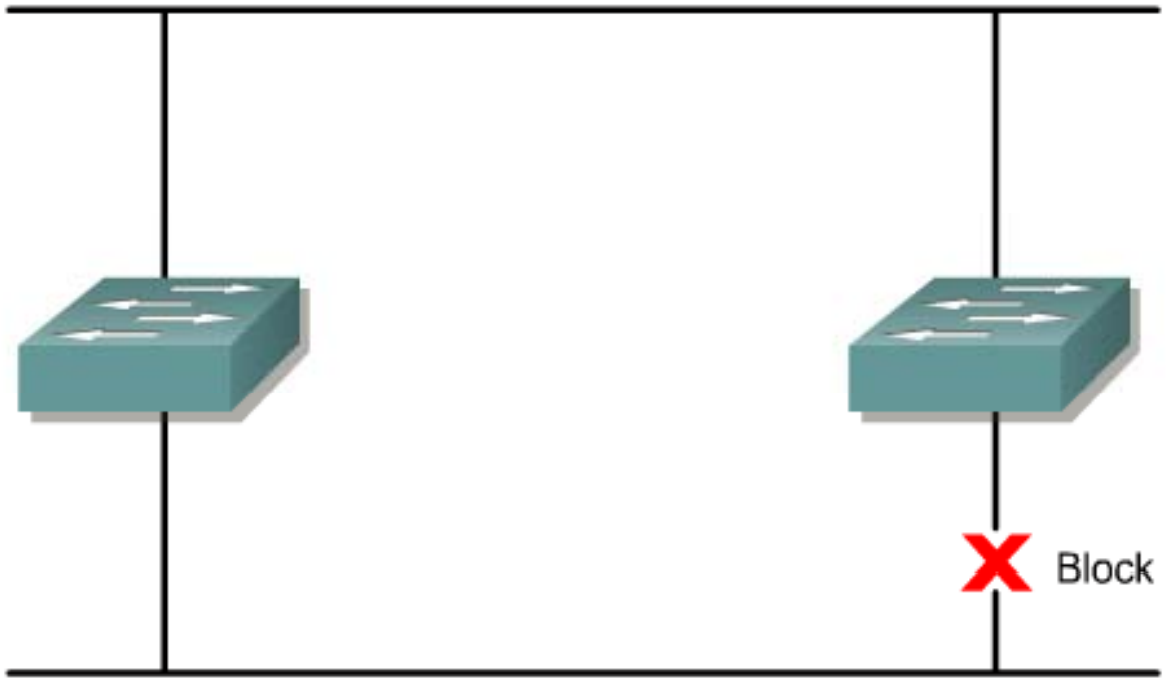


Minimize the total length of wire connecting the customers

Spanning Tree Protocol

Motivation

STP: Redundancy Without Loops



Provides a loop-free, redundant network topology by placing certain ports in the blocking state.

Spanning Tree Protocol (STP)

- The algorithm used to create this loop free logical topology is the spanning-tree algorithm.
- Allows redundancy without loops
- The loop free logical topology created is called a tree.
- This topology is a star or extended star logical topology, the spanning tree of the network.
- It is a spanning tree because all devices in the network are reachable or spanned.
- Slow to converge, so now there is Rapid Spanning Tree

STP

- Ethernet switches can implement the IEEE 802.1D Spanning-Tree Protocol and use the spanning-tree algorithm to construct a loop free shortest path network.
- Shortest path is based on cumulative link costs. Link costs are based on the speed of the link.
- STP establishes a root node, called the root bridge. The Spanning-Tree Protocol constructs a topology that has one path for reaching every network node.
- The resulting tree originates from the root bridge.
- Redundant links that are not part of the shortest path tree are blocked, causing a loop free topology

Spanning Tree Link Costs

Link Speed	Cost(Revised IEEE Spec)	Cost (Previous IEEE Spec)
10 Gbps	2	1
1 Gbps	4	1
100 Mbps	19	10
10 Mbps	100	100

Spanning tree operation

Motivation

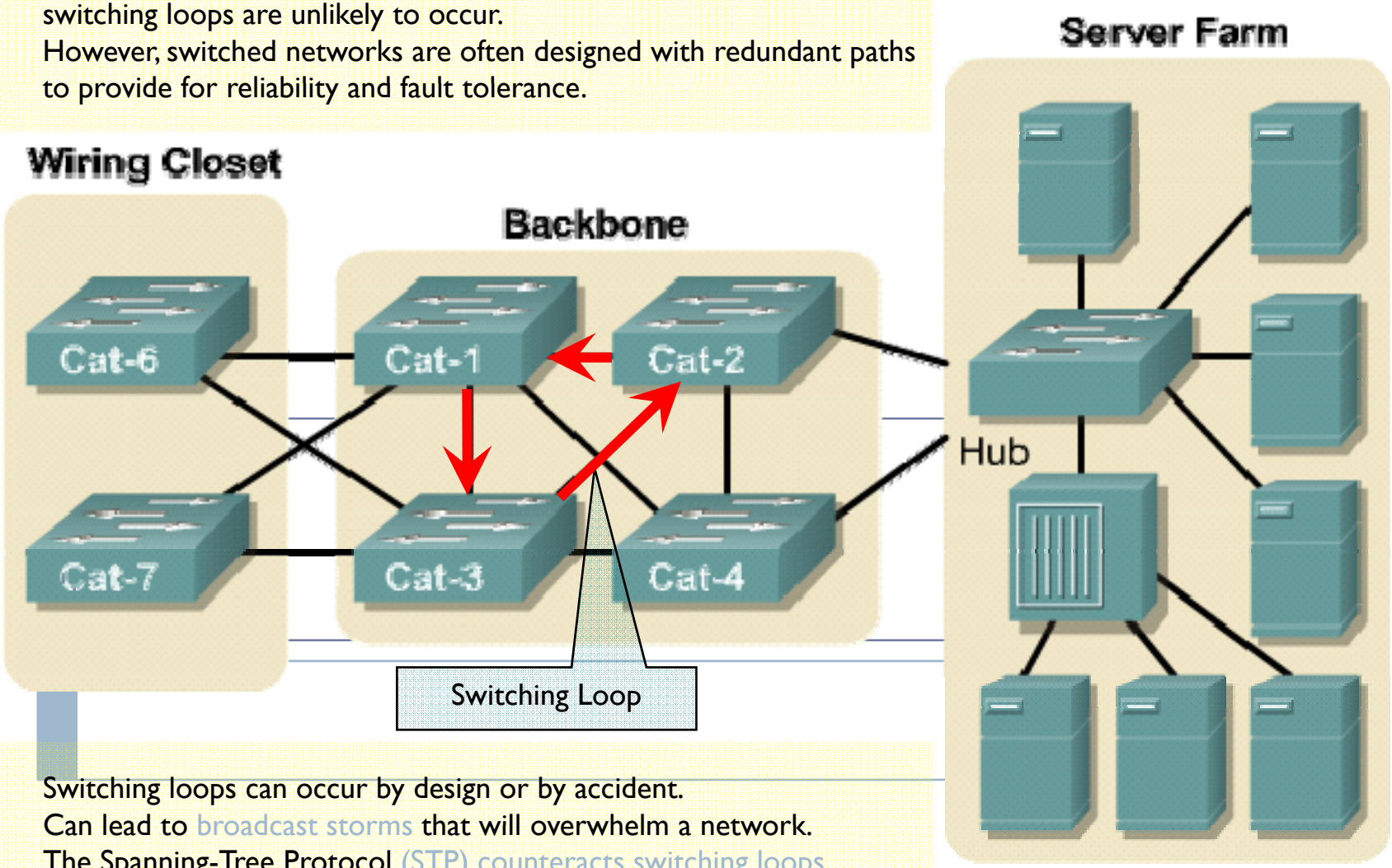
Spanning-Tree Operation

FIGURES

1

2

- When multiple switches are arranged in a simple hierarchical tree, switching loops are unlikely to occur.
- However, switched networks are often designed with redundant paths to provide for reliability and fault tolerance.



- Switching loops can occur by design or by accident.
- Can lead to broadcast storms that will overwhelm a network.
- The Spanning-Tree Protocol (STP) counteracts switching loops.

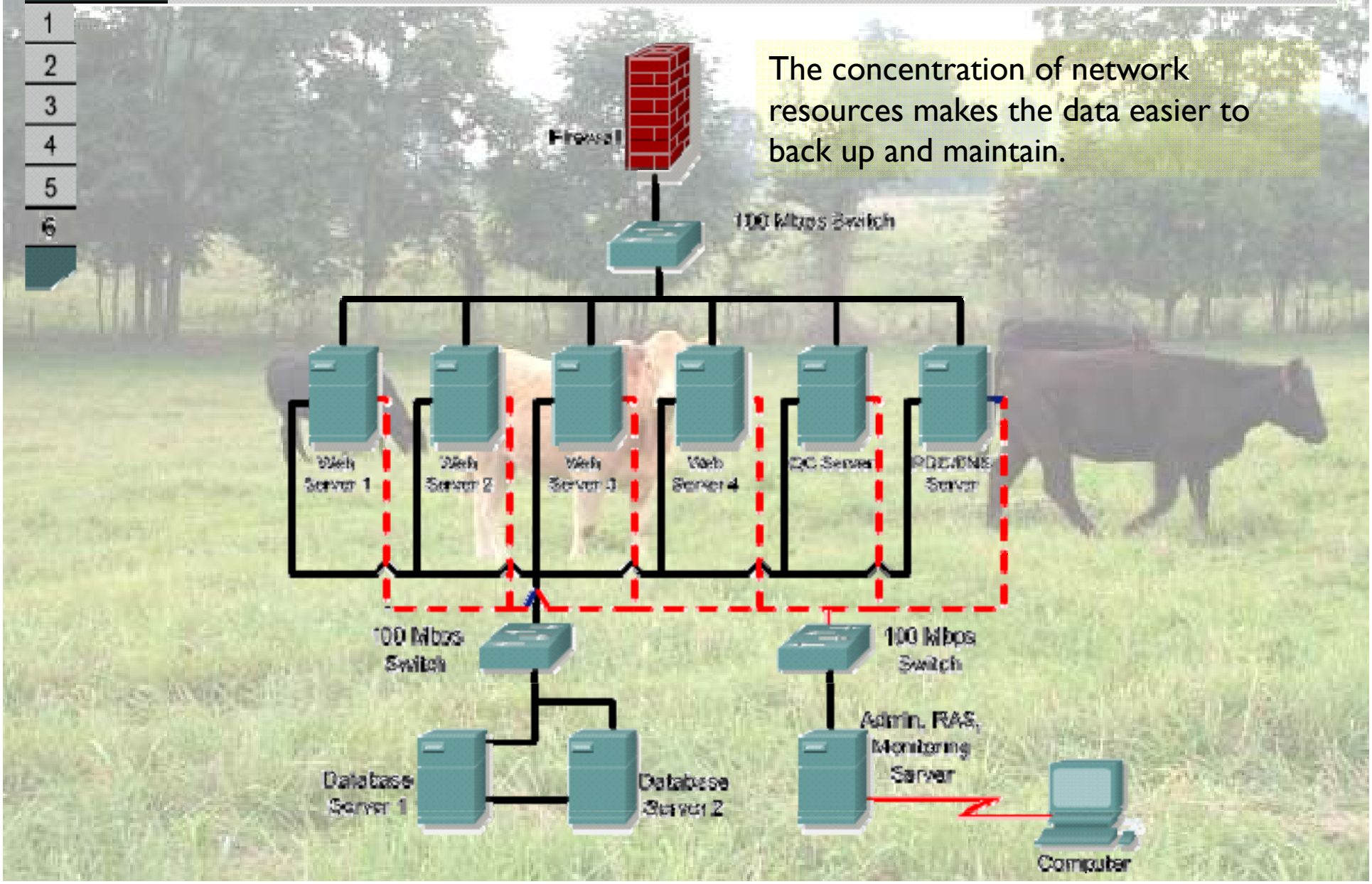
Server Farm

FIGURES

1 2 3 4 5 6

8.1.6 Spanning-Tree Protocol

The concentration of network resources makes the data easier to back up and maintain.



STP States

- Each switch in a LAN using STP sends special messages called Bridge Protocol Data Units (BPDUs)
- This tells other switches that it exists, and is used to elect a root bridge for the network.
- The switches then use the Spanning-Tree Algorithm (STA) to resolve and shut down redundant paths.
- STP creates a logical hierarchical tree with no loops.
- However, the alternate paths are still available should they be needed.

States	Purpose
Blocking	Receives BPDUs only
Listening	Building "active" topology
Learning	Building bridging table
Forwarding	Sending and receiving user data
Disabled	Administratively down

You must know the five STP states and their purpose.

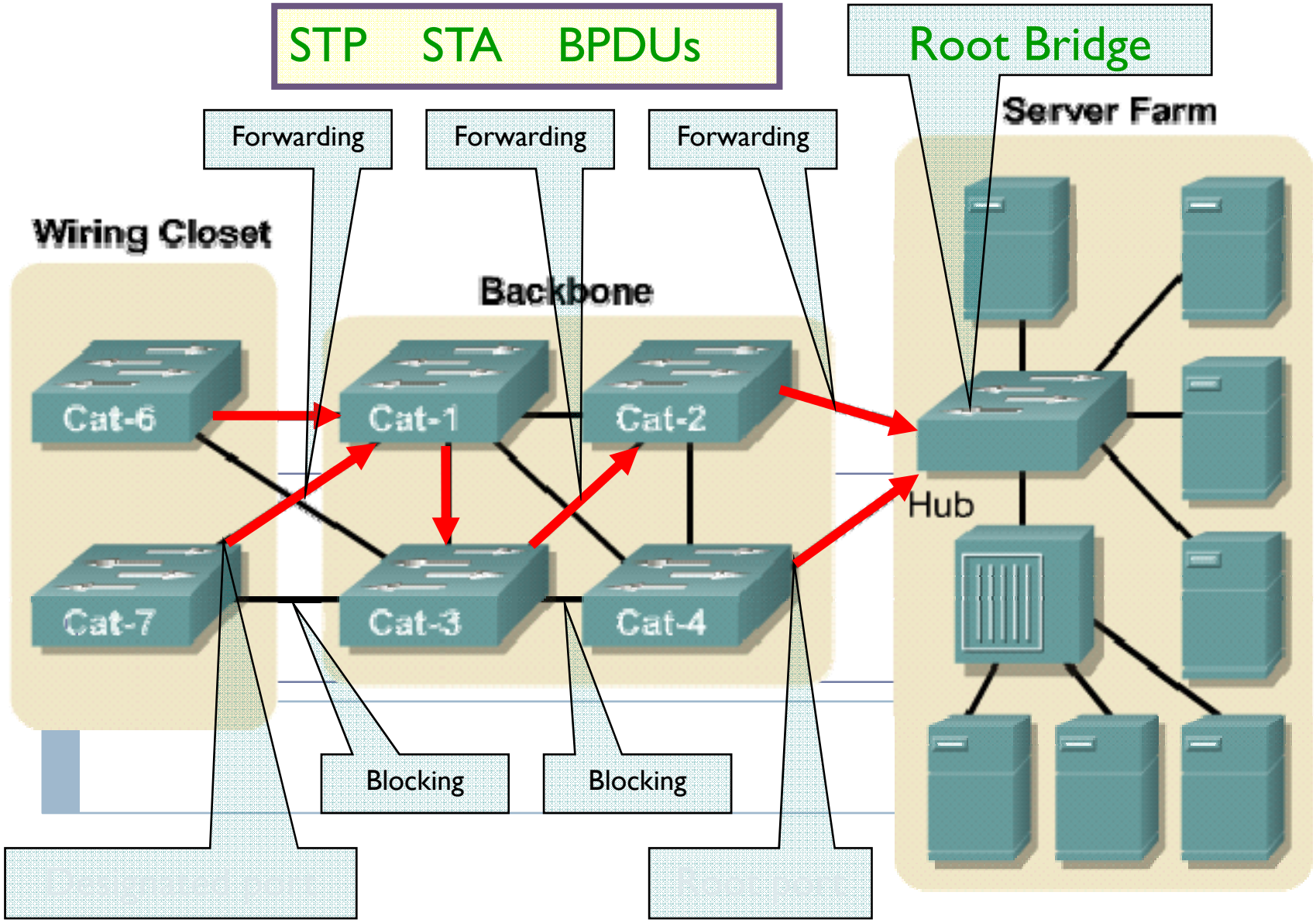
Spanning-Tree Operation

FIGURES

8.1.6 Spanning-Tree Protocol

1

2



Why minimum spanning trees?

- The standard application is to a problem like phone network design.
- You have a business with several offices; you want to lease phone lines to connect them up with each other; and the phone company charges different amounts of money to connect different pairs of cities.
- You want a set of lines that connects all your offices with a minimum total cost.
- It should be a spanning tree, since if a network isn't a tree you can always remove some edges and save money

Solving TSP using Spanning tree

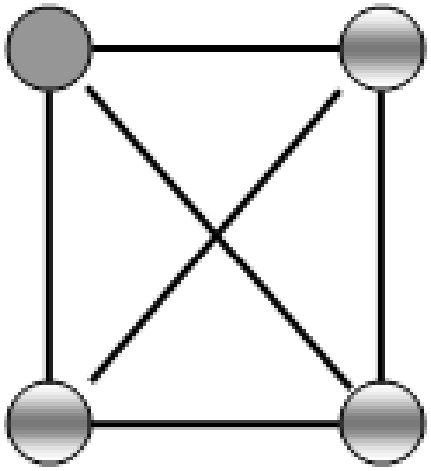
- A less obvious application is that the minimum spanning tree can be used to **approximately solve** the traveling salesman problem.
- A convenient formal way of defining this problem is to find the shortest path that visits each point at least once.
- Note that if you have a path visiting all points exactly once, it's a special kind of tree.
- For instance in the example above, twelve of sixteen spanning trees are actually paths.
- If you have a path visiting some vertices more than once, you can always drop some edges to get a tree.
- So in general the MST weight is less than the TSP weight, because it's a minimization over a strictly larger set

Solving TSP using Spanning tree

- On the other hand, if you draw a path tracing around the minimum spanning tree, you trace each edge twice and visit all points, so the TSP weight is less than twice the MST weight.
- Therefore this tour is within a factor of two of optimal. There is a more complicated way (*Christofides' heuristic*) of using minimum spanning trees to find a tour within a factor of 1.5 of optimal;

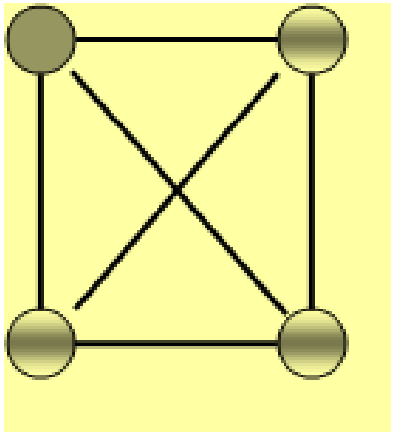
Spanning Tree

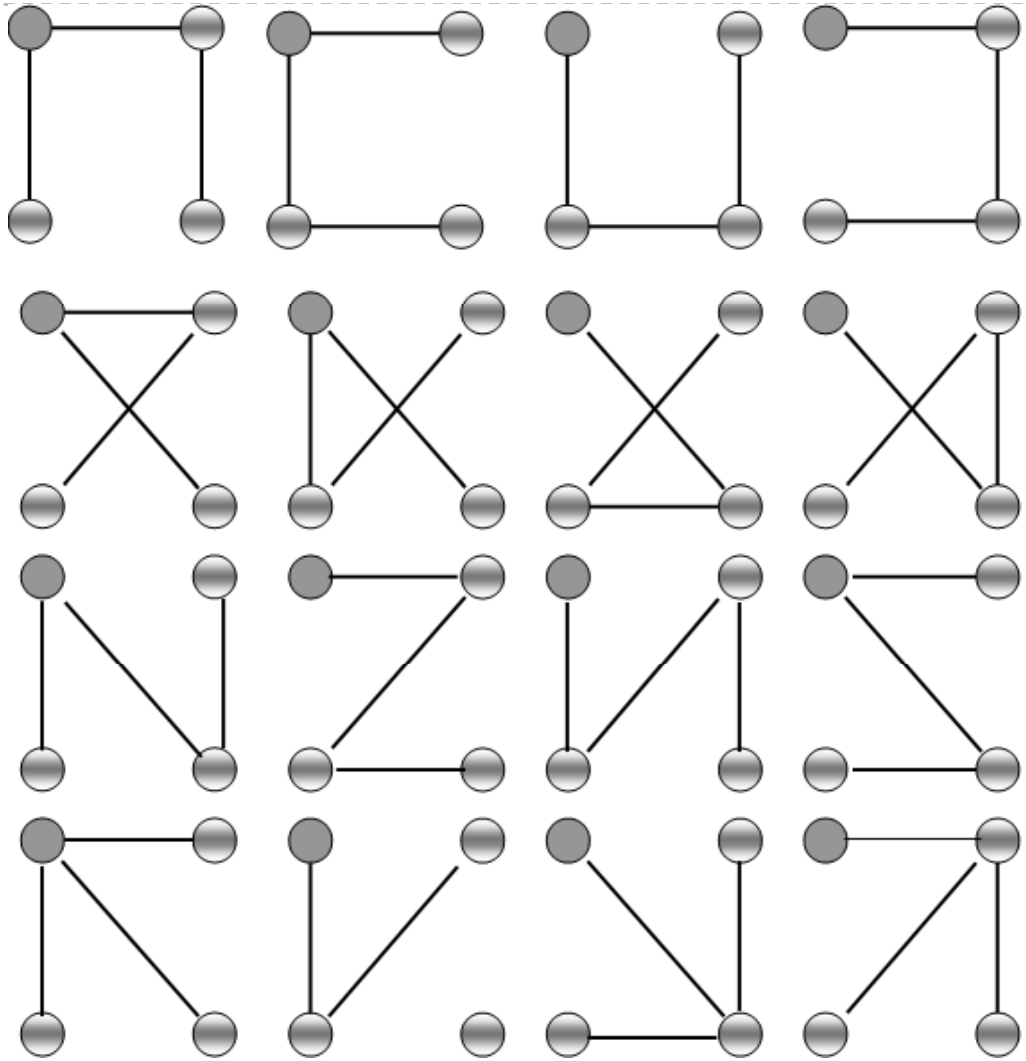
- A *spanning tree* of a graph is just a sub-graph that contains all the vertices and is a tree.



A graph may have many spanning trees; for instance the complete graph on four vertices has sixteen spanning trees:

Complete graph on 4 vertices has 16 spanning trees





Minimum Spanning Trees

- Suppose that the edges of the graph have weights or lengths. The weight of a tree will be the sum of weights of its edges.
- Based on the example, we can see that different trees have different lengths.
- The question is: how to find the minimum length spanning tree?

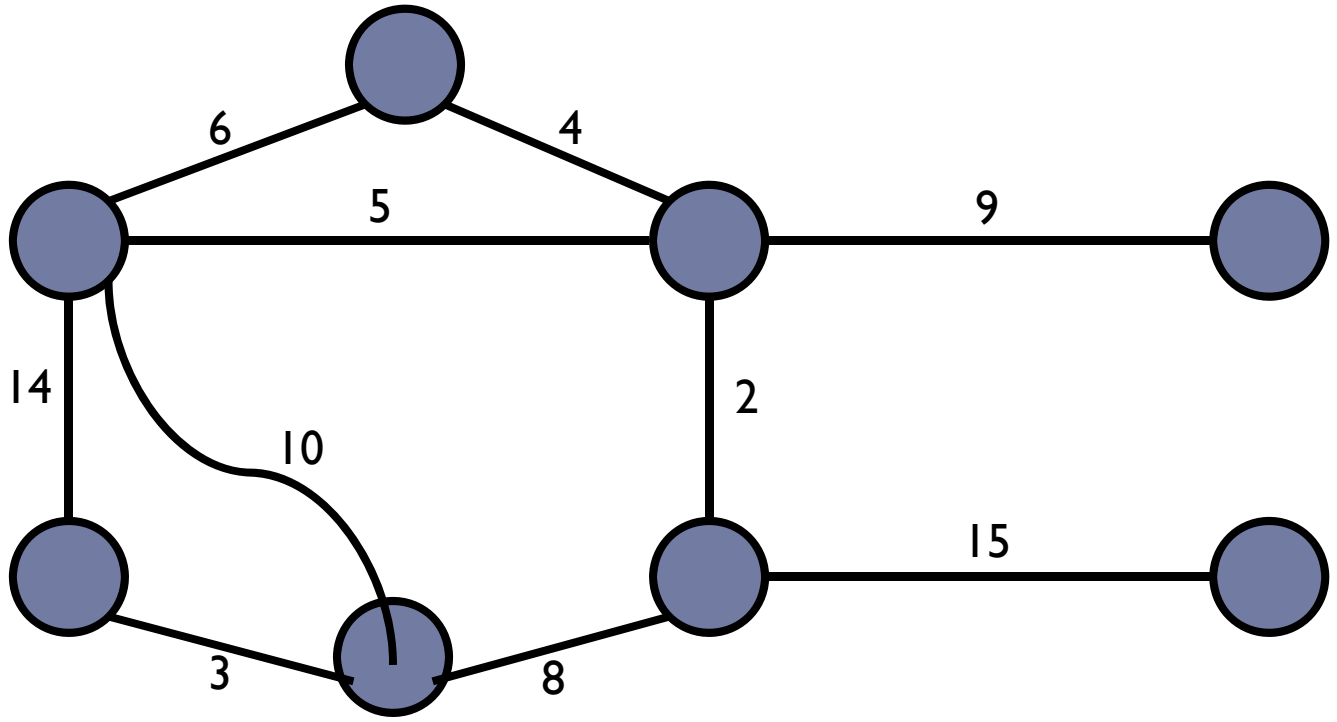
Minimum Spanning Tree

- A **spanning tree** of a graph is just a subgraph that contains all the vertices and is a **tree**.

- The **Minimum Spanning Tree** for a given graph is the**Spanning Tree** of **minimum** cost for that graph. On a **weighted** graph, A MST: connects all vertices through edges with least **weights**.

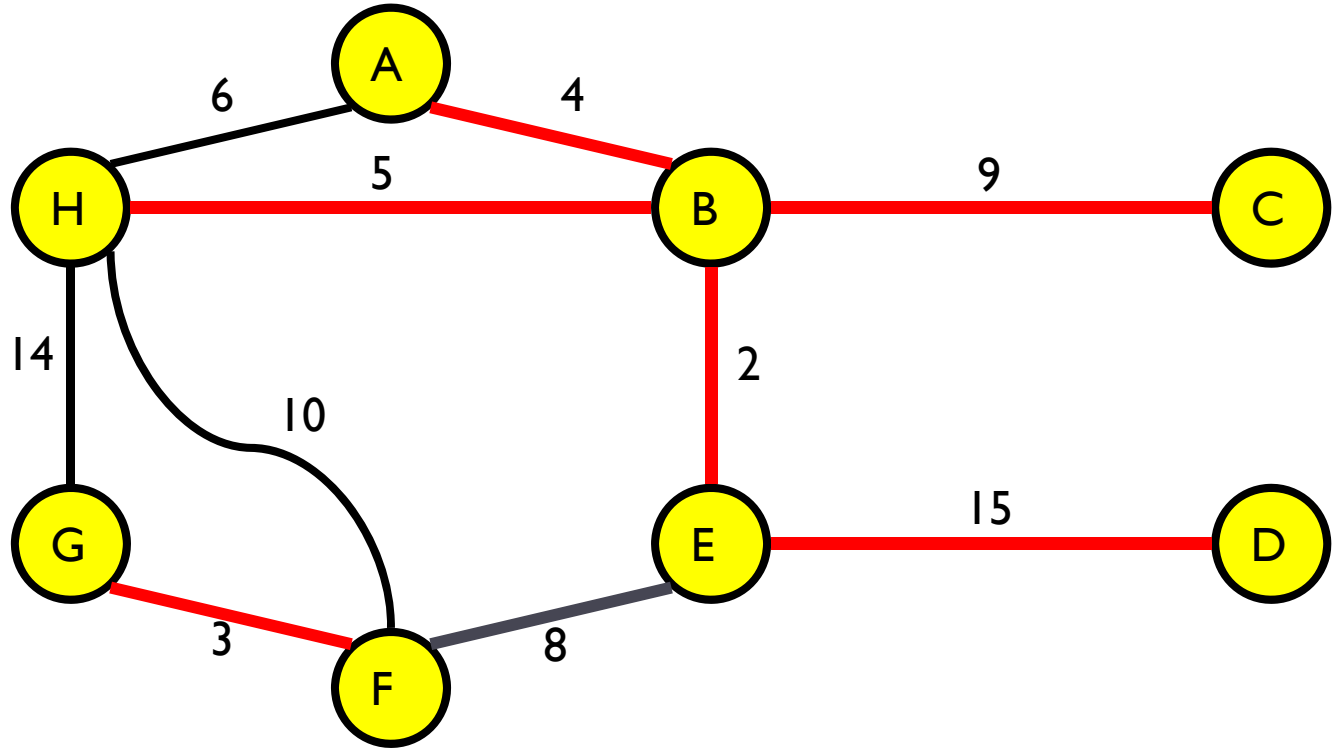
Example: Minimum Spanning Tree

- Problem: given a connected, undirected, weighted graph, find a *spanning tree* using edges that minimize the total weight

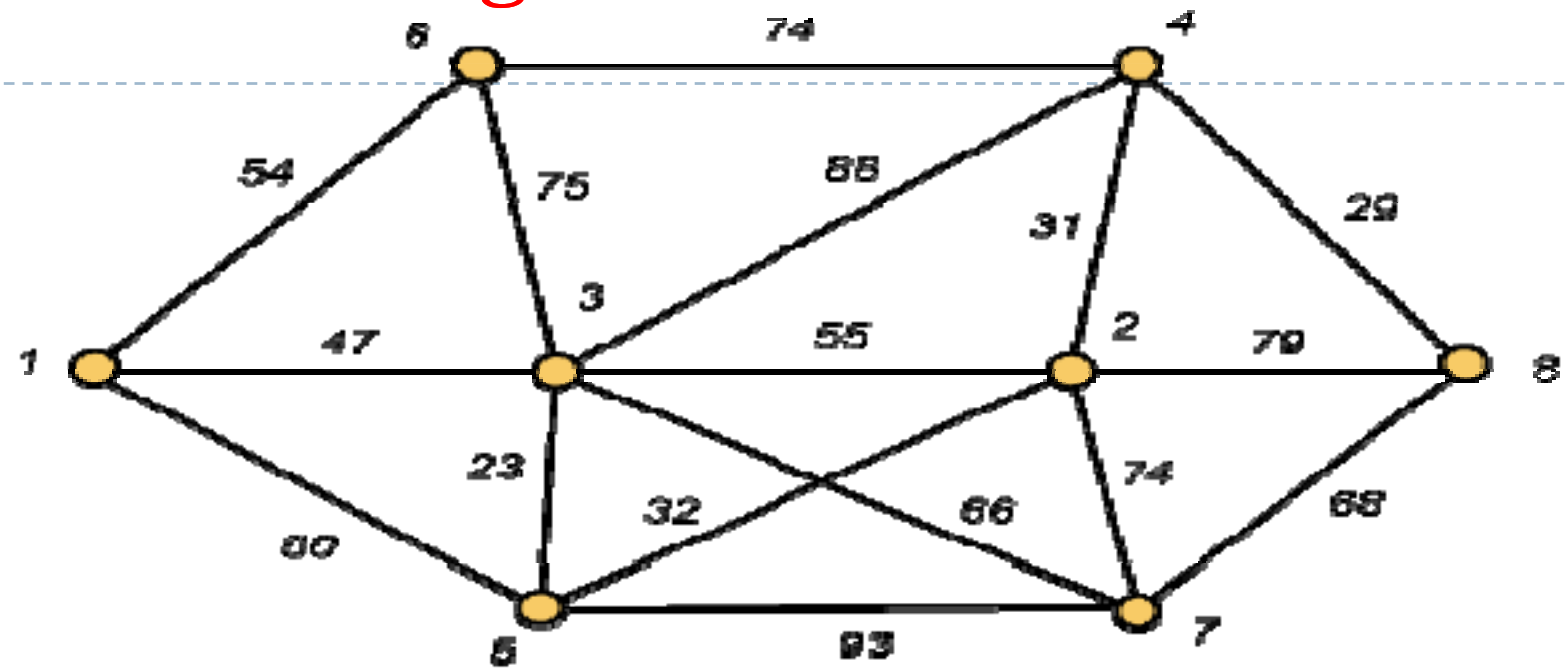


Example: Minimum Spanning Tree

- Answer:

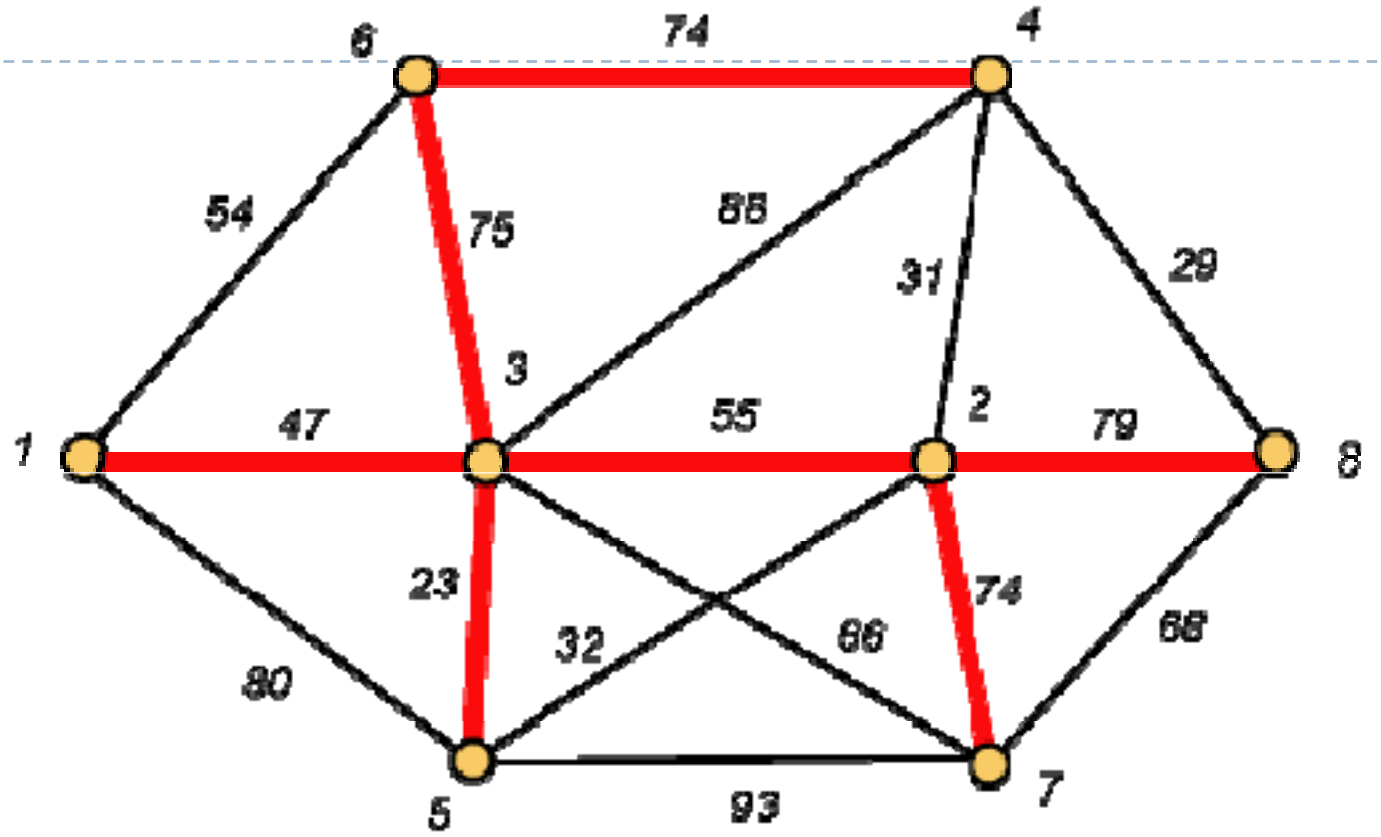


A Networking Problem



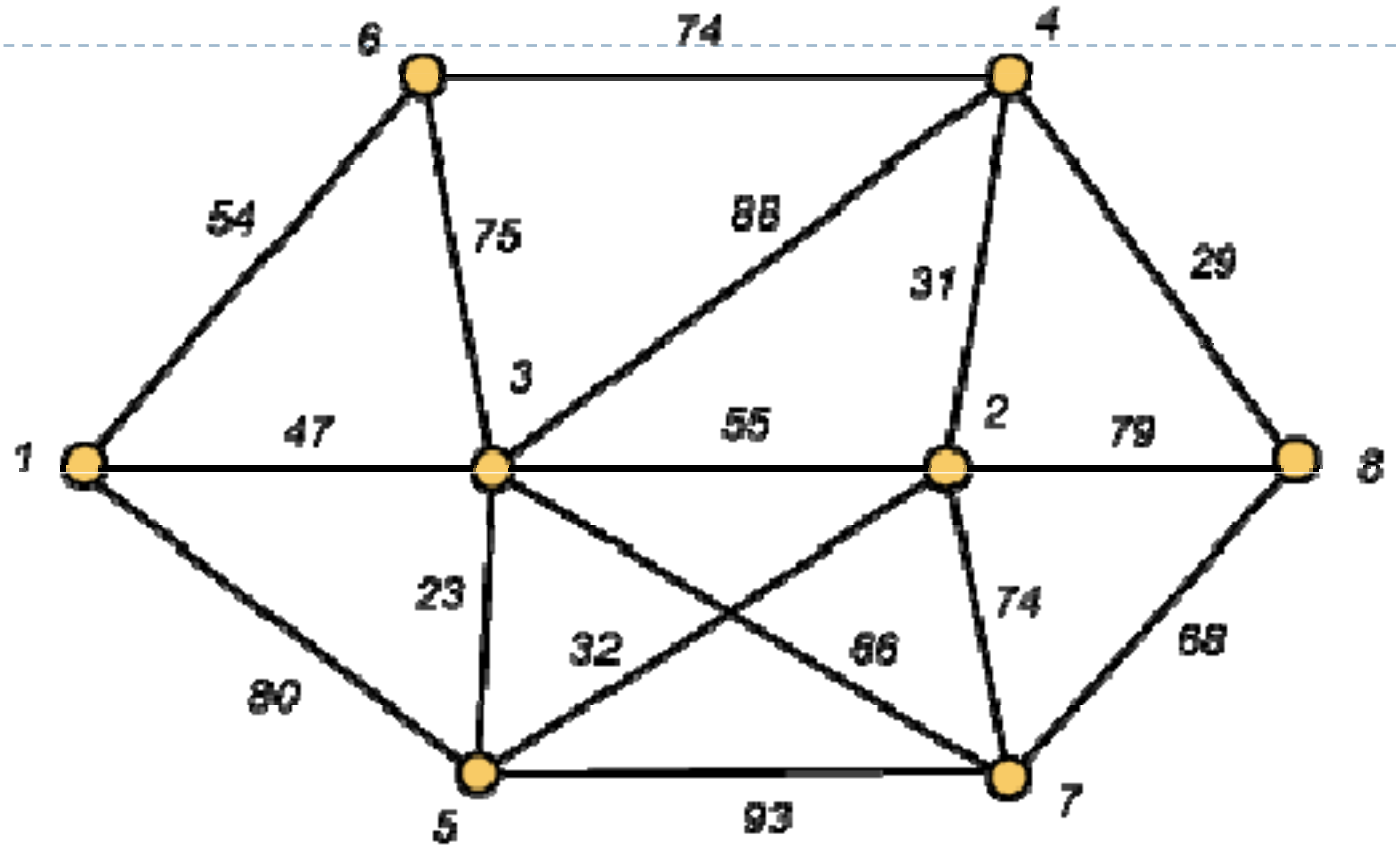
Problem: The vertices represent 8 regional data centers which need to be connected with high-speed data lines. Feasibility studies show that the links illustrated above are possible, and the cost in millions of dollars is shown next to the link. Which links should be constructed to enable full communication (with relays allowed) and keep the total cost minimal.

Links Will Form a Spanning Tree



Cost (T) = 47 + 23 + 75 + 74 + 55 + 74 + 79 = 427

Minimum Weight Spanning Trees



Problem: Given a connected graph with non-negative weights on the edges, find a spanning tree T for which the sum of the weights on the edges in T is as small as possible.

Why Not Try *All* Possibilities?

- Suppose the graph has n vertices. Then the number of possible spanning trees can be as large as n^{n-2}.
- When $n = 75$, this means that the number of spanning trees can be as large as

75765628046446014790863186515904134648140 67\
83308840339247043281018024279971356804708193\
521946668624877929 6875

- Lemma: Let X be any subset of the vertices of G, and let edge e be the smallest edge connecting X to G-X. Then e is part of the minimum spanning tree.
- Proof: Suppose you have a tree T not containing e; then I want to show that T is not the MST. Let e=(u,v), with u in X and v not in X. Then because T is a spanning tree it contains a unique path from u to v, which together with e forms a cycle in G. This path has to include another edge f connecting X to G-X. T+e-f is another spanning tree (it has the same number of edges, and remains connected since you can replace any path containing f by one going the other way around the cycle). It has smaller weight than t since e has smaller weight than f. So T was not minimum, which is what we wanted to prove

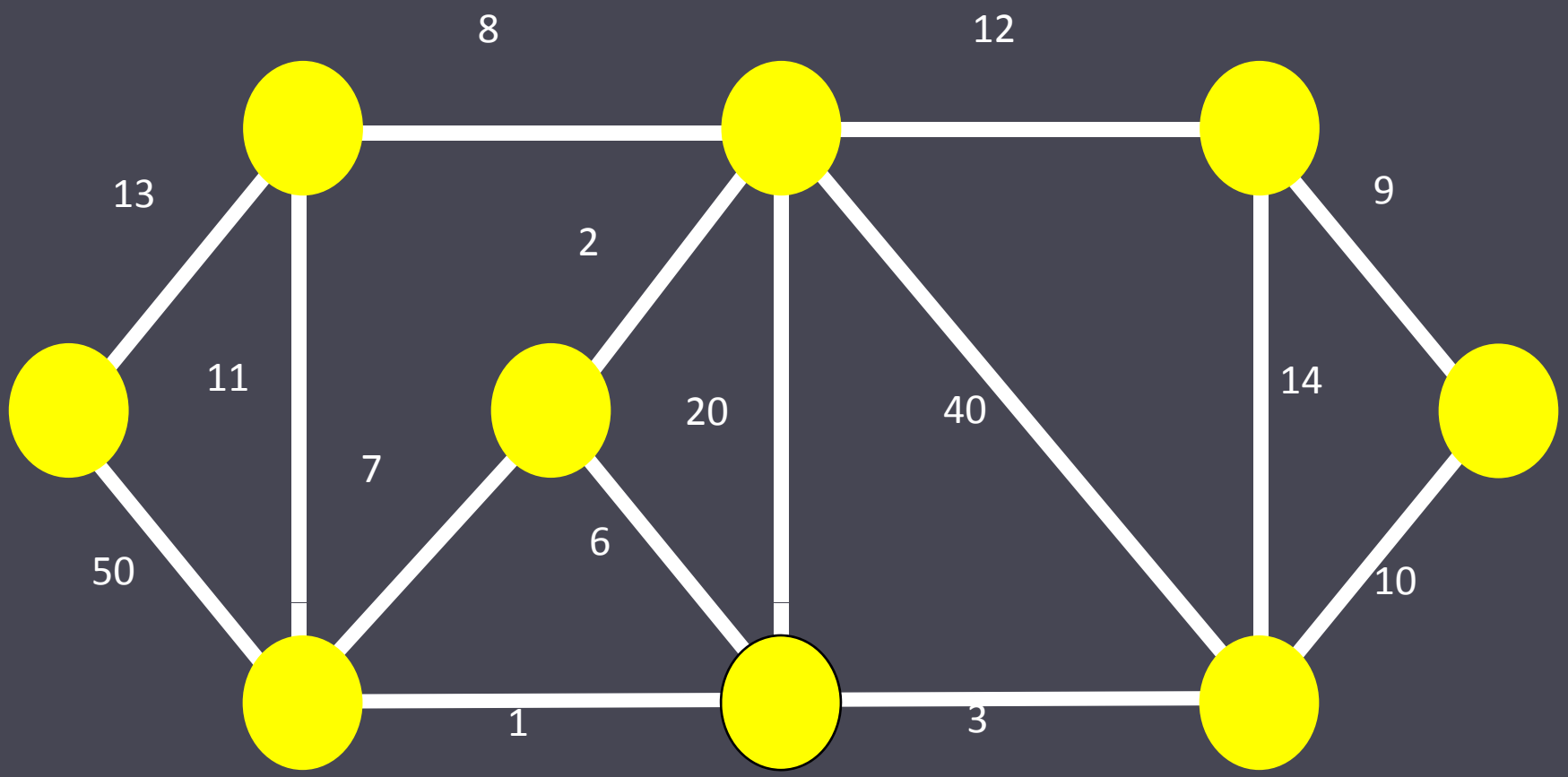
Some History

- **Borůvka** [1926] : First algorithm
  - for electrical coverage of Moravia
- **Kruskal** [1956] : Kruskal's algorithm
- **Jarník** [1930],
- **Prim** [1957] : Prim's algorithm
- **Fredman-Tarjan** [1987] : O(E log*(V)) time
- **Gabow et al** [1986]: O(E log log*(V) time
- **Chazelle** [1999]: O(E α(E,V)) time
- **Remark: log* = iterated log, α(m,n) = inverse Ackermann**

Minimum Spanning Trees

- The three classical minimum-spanning tree algorithms :
  - Boruvka's Algorithm
  - Kruskal's Algorithm
  - Prim's Algorithm

Kruskal's Algorithm



Kruskal's algorithm

1. sort the edges of G in increasing order by length keep a subgraph S of G, initially empty
2. for each edge e in sorted order
3. if the endpoints of e are disconnected in S add e to S
4. return S

- Note that, whenever you add an edge (u,v), it's always the smallest connecting the part of S reachable from u with the rest of G, so by the lemma it must be part of the MST

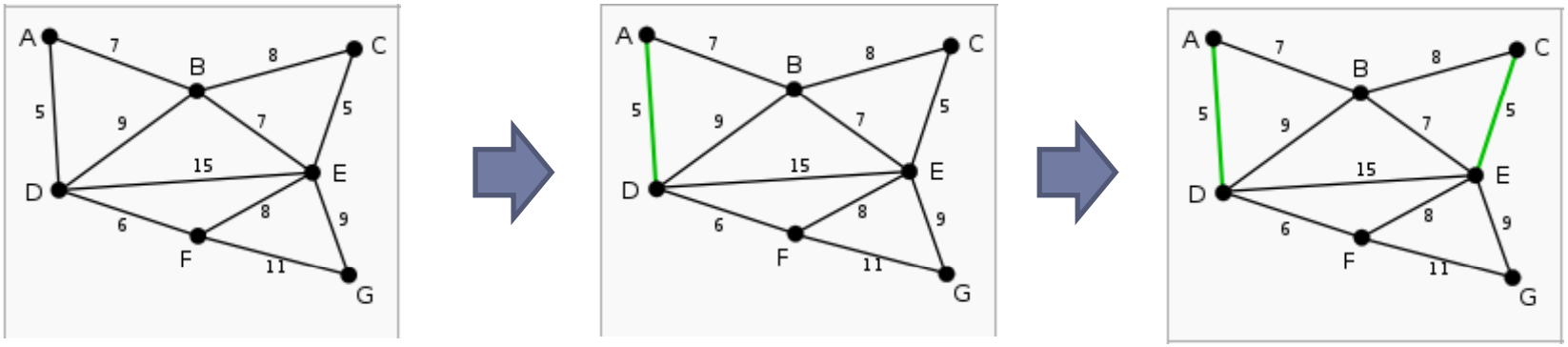
Kruskal's Algorithm

- **Joseph Bernard Kruskal, Jr**
- **Kruskal Approach:**
  - Select the minimum weight edge that does not form a cycle

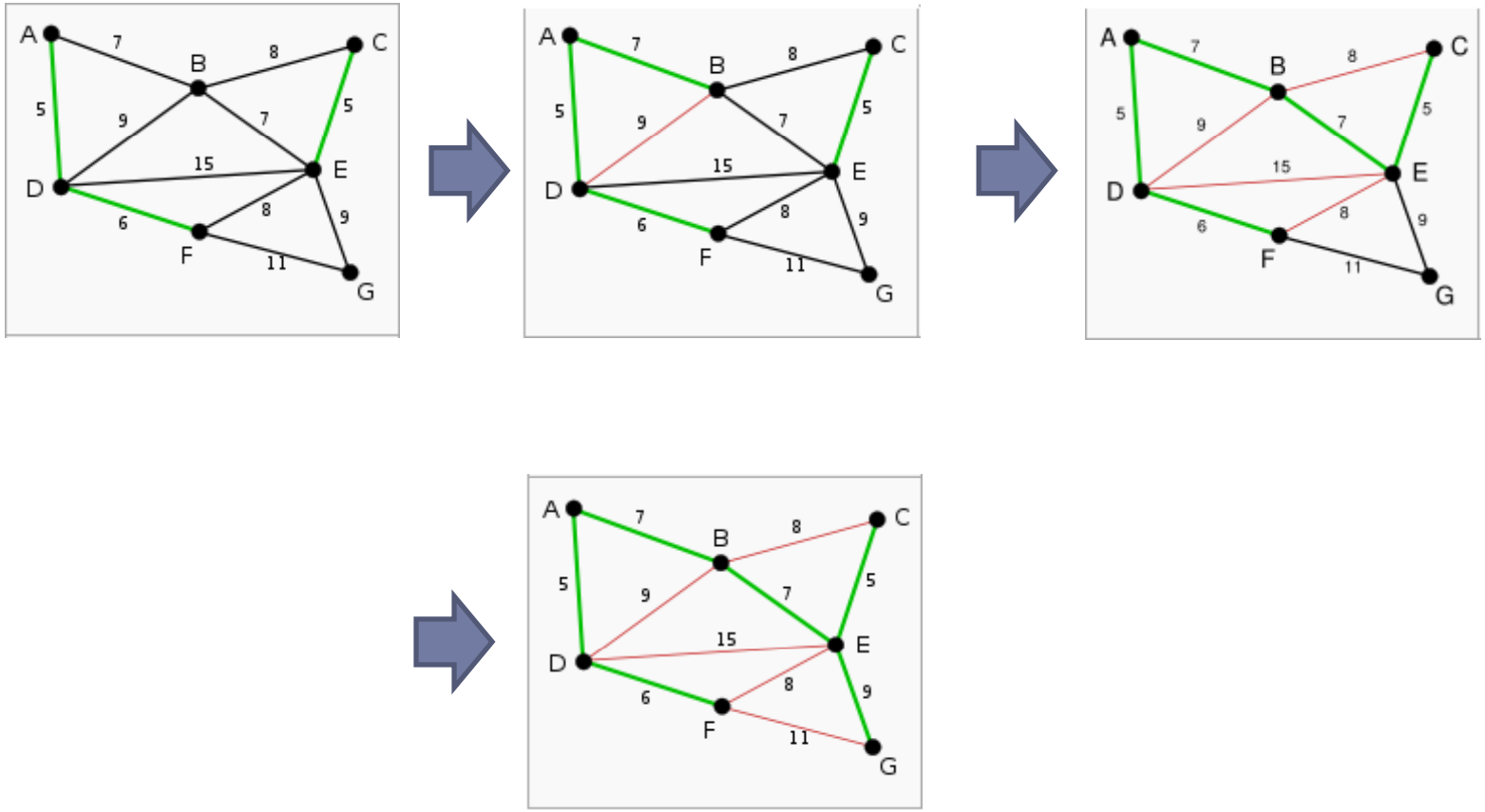
Kruskal's Algorithm:

```
sort the edges of G in increasing order by length
keep a subgraph S of G, initially empty
for each edge e in sorted order
     if the endpoints of e are disconnected in S
     add e to S
 return S
```

Kruskal's Algorithm - Example



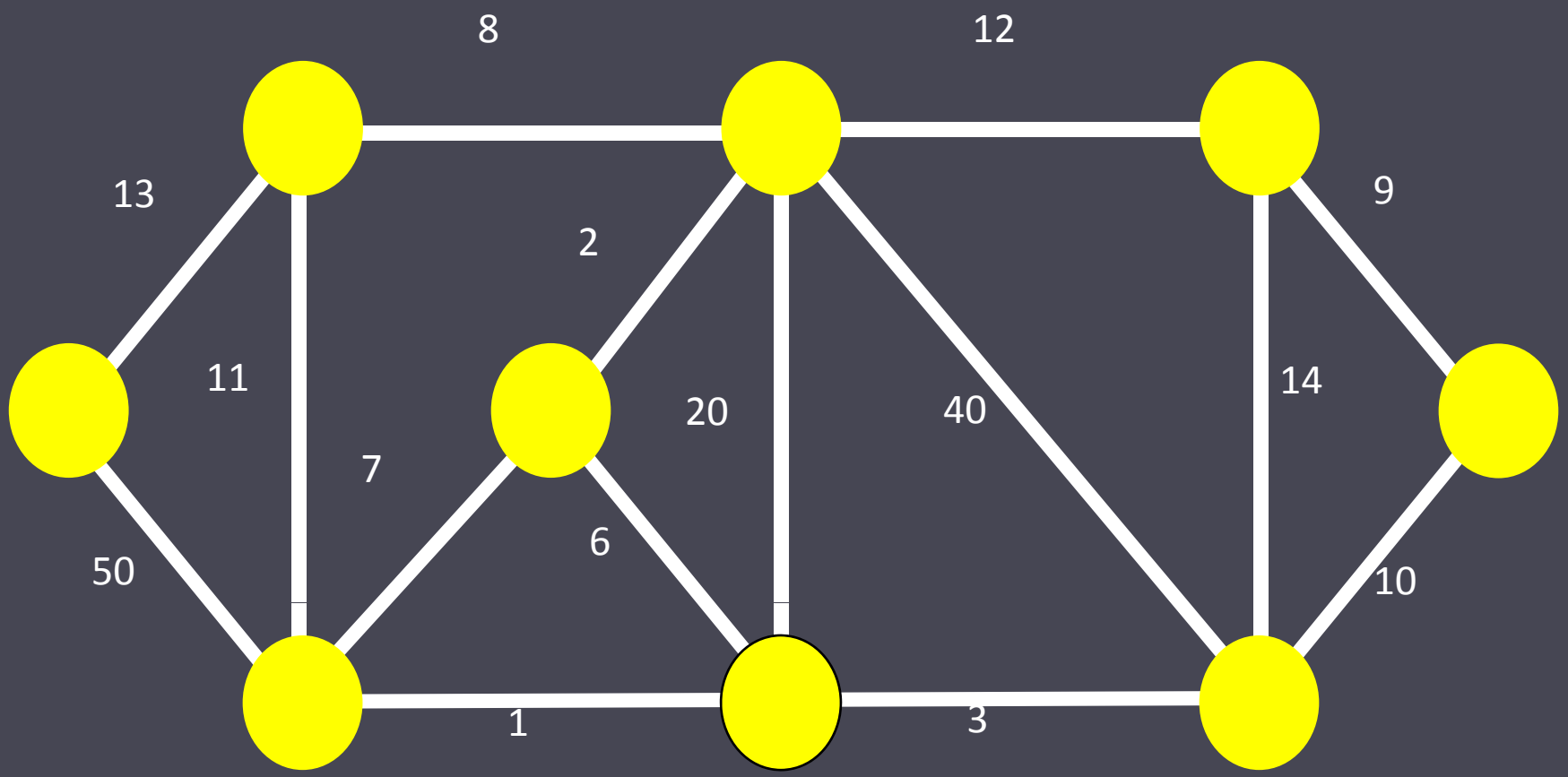
Kruskal's Algorithm - Example



Kruskal's algorithm

```
1    Algorithm Kruskal(E, cost, n, t)
2    // E is the set of edges in G. G has n vertices. cost[u, v] is the
3    // cost of edge (u, v). t is the set of edges in the minimum-cost
4    // spanning tree. The final cost is returned.
5    {
6        Construct a heap out of the edge costs using Heapify;
7        for i := 1 to n do parent[i] := -1;
8        // Each vertex is in a different set.
9        i := 0; mincost := 0.0;
10       while ((i < n - 1)  and (heap not empty)) do
11       {
12           Delete a minimum cost edge (u, v) from the heap
13           and reheapify using Adjust;
14           j := Find(u); k := Find(v);
15           if (j ≠ k) then
16           {
17               i := i + 1;
18               t[i, 1] := u; t[i, 2] := v;
19               mincost := mincost + cost[u, v];
20               Union(j, k);
21           }
22       }
23       if (i ≠ n - 1) then write ("No spanning tree");
24       else return mincost;
25   }
```

Prim's algorithm



Prim's algorithm

1. let T be a single vertex x
2. while (T has fewer than n vertices)
3. {
4. find the smallest edge connecting T to G-T
   add it to T
5. }

- Since each edge added is the smallest connecting T to G-T, the lemma we proved shows that we only add edges that should be part of the MST

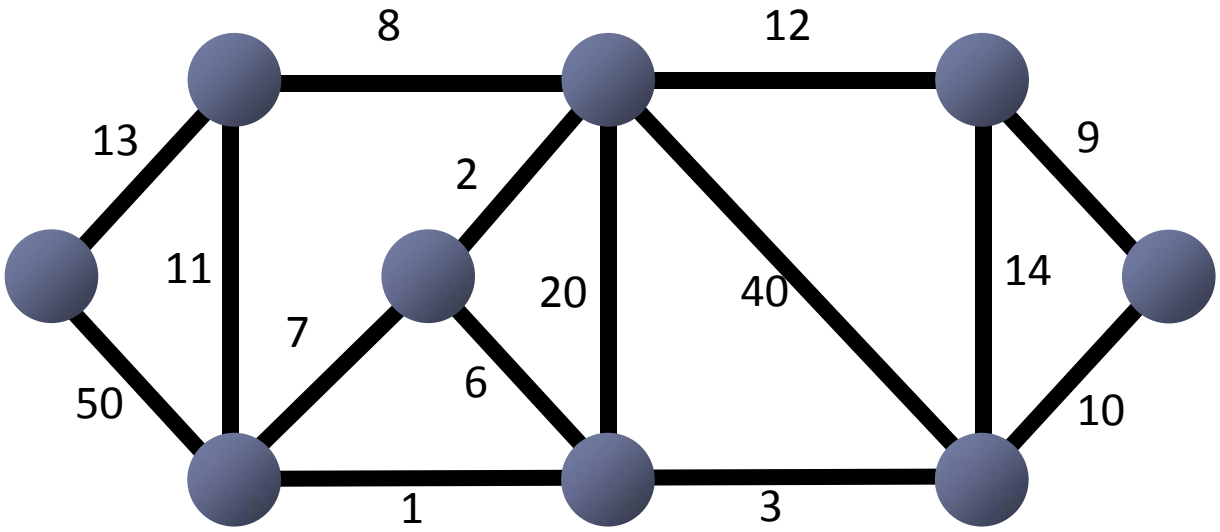
Prim's Algorithm

- **Robert Clay Prim**
- Prim Approach:
  - Choose an arbitrary start node v
  - At any point in time, we have connected component N containing v and other nodes V-N
  - Choose the minimum weight edge from N to V-N

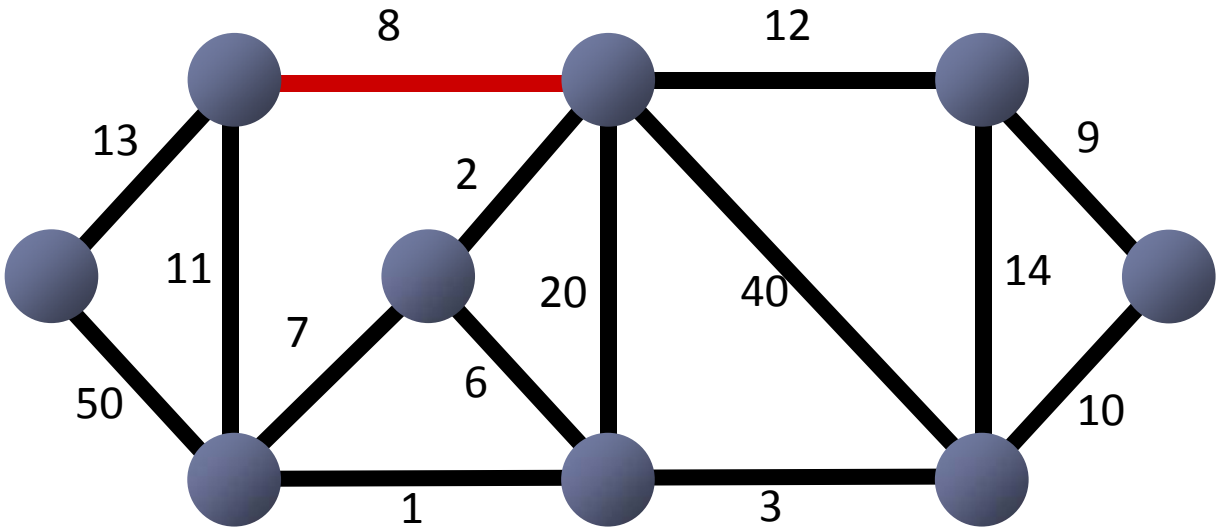
Prim's Algorithm:

```
let T be a single vertex x
while (T has fewer than n vertices)
{
    find the smallest edge connecting T to G-T
    add it to T
}
```

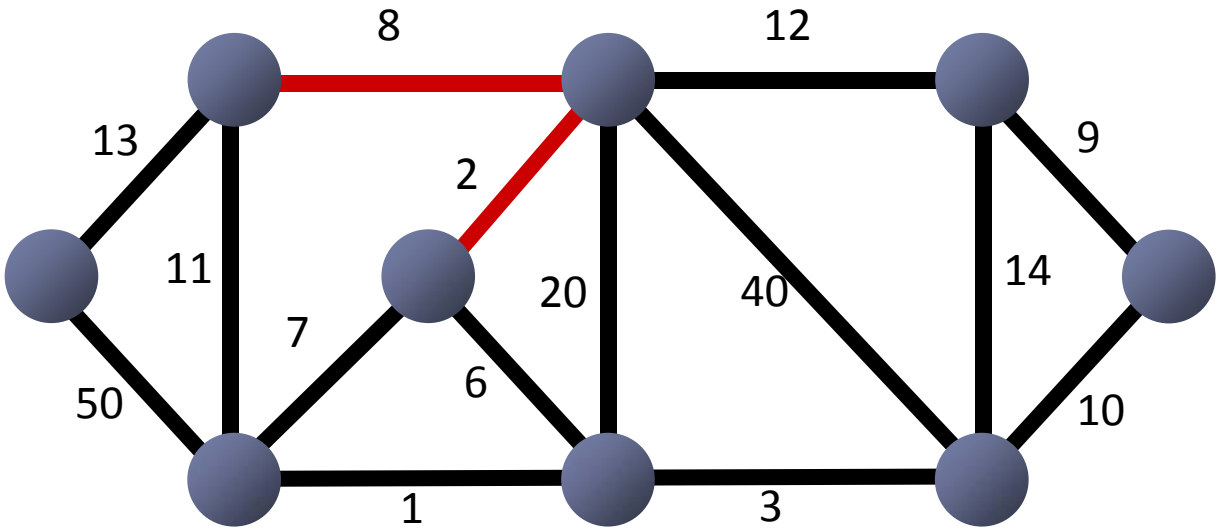
Prim's Algorithm - Example



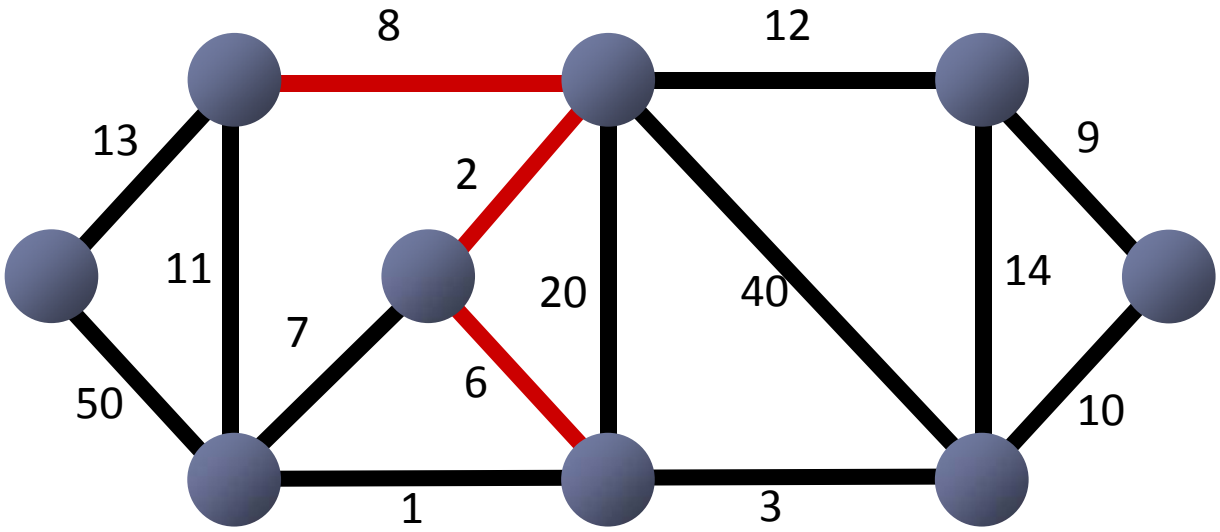
Prim's Algorithm - Example



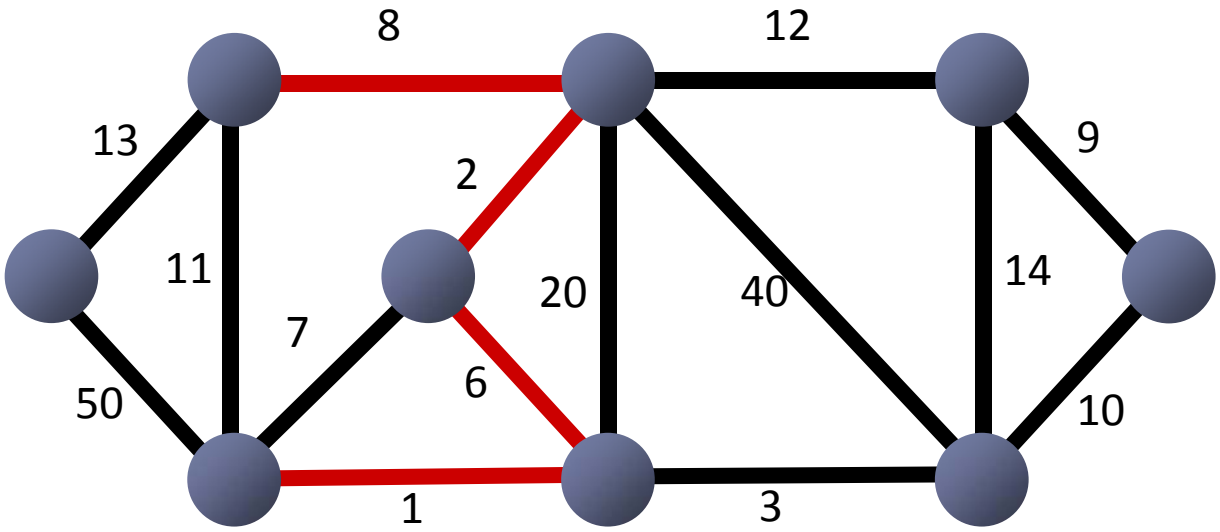
Prim's Algorithm - Example



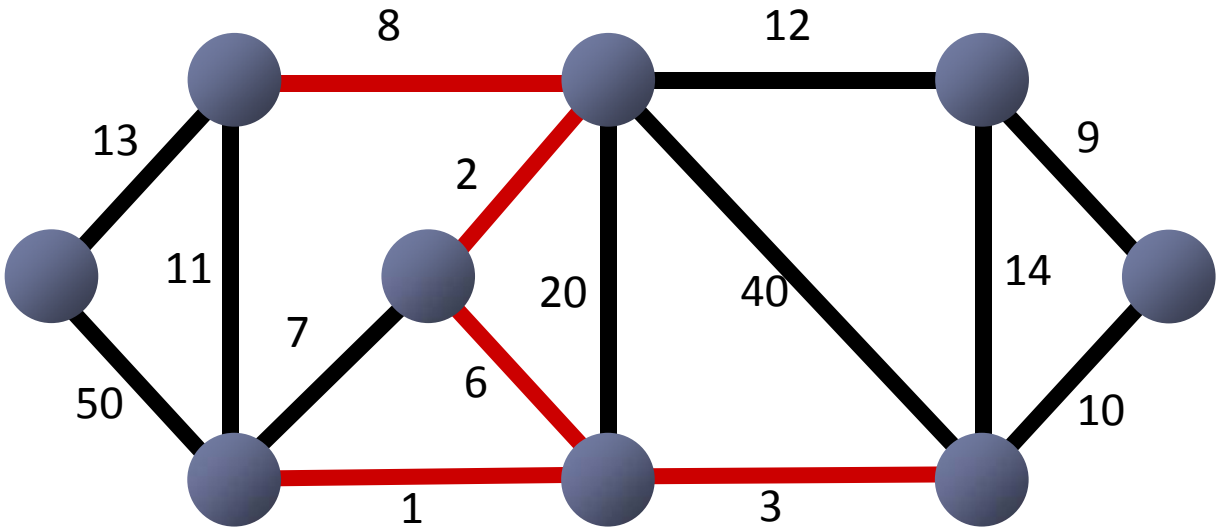
Prim's Algorithm - Example



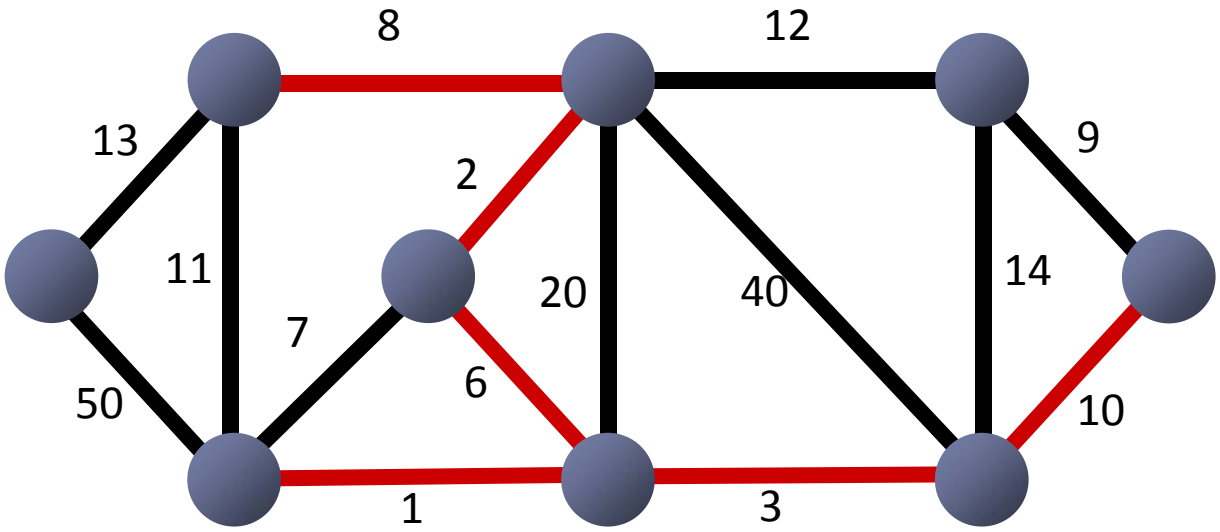
Prim's Algorithm - Example



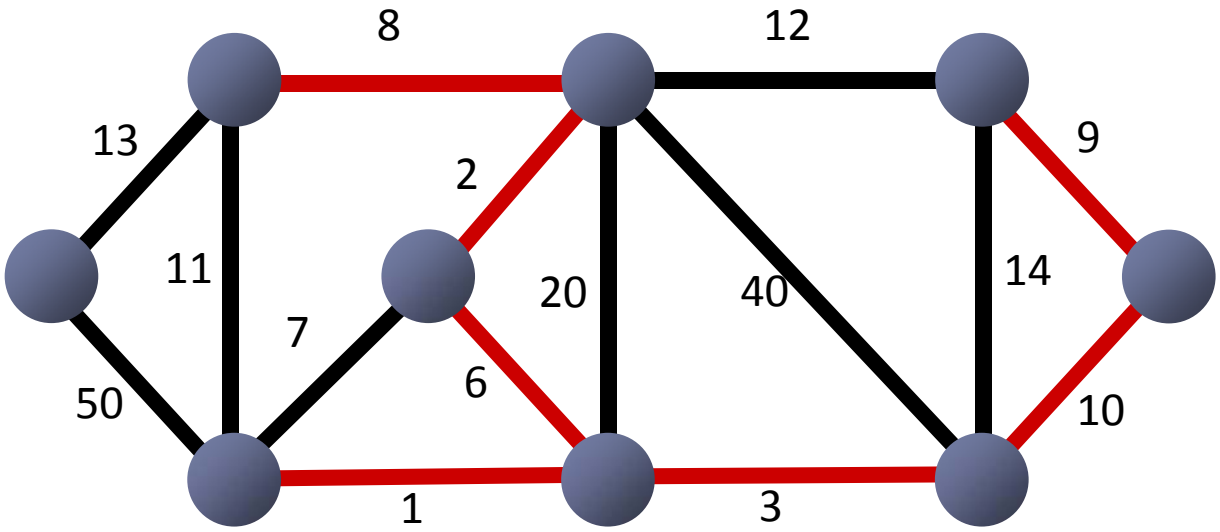
Prim's Algorithm - Example



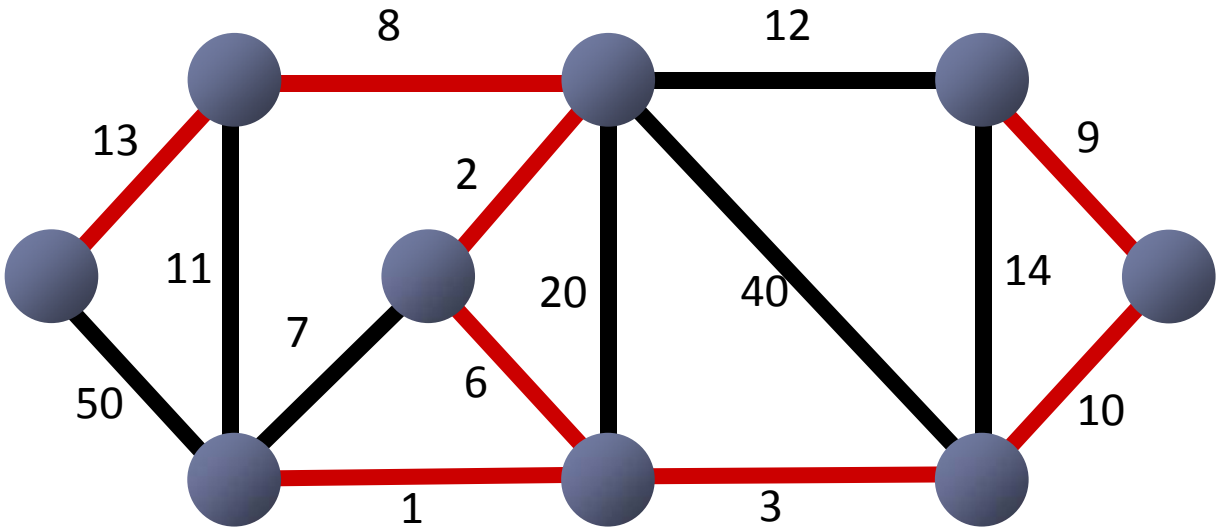
Prim's Algorithm - Example



Prim's Algorithm - Example



Prim's Algorithm - Example



Prim's algorithm

```
1   Algorithm Prim(E, cost, n, t)
2   // E is the set of edges in G. cost[1 : n, 1 : n] is the cost
3   // adjacency matrix of an n vertex graph such that cost[i, j] is
4   // either a positive real number or ∞ if no edge (i, j) exists.
5   // A minimum spanning tree is computed and stored as a set of
6   // edges in the array t[1 : n − 1, 1 : 2]. (t[i, 1], t[i, 2]) is an edge in
7   // the minimum-cost spanning tree. The final cost is returned.
8   {
9       Let (k, l) be an edge of minimum cost in E;
10      mincost := cost[k, l];
11      t[1, 1] := k; t[1, 2] := l;
12      for i := 1 to n do  // Initialize near.
13          if (cost[i, l] < cost[i, k]) then near[i] := l;
14          else near[i] := k;
15      near[k] := near[l] := 0;
16      for i := 2 to n − 1 do
17      { // Find n − 2 additional edges for t.
18          Let j be an index such that near[j] ≠ 0 and
19          cost[j, near[j]] is minimum;
20          t[i, 1] := j; t[i, 2] := near[j];
21          mincost := mincost + cost[j, near[j]];
22          near[j] := 0;
23          for k := 1 to n do // Update near[ ].
24              if ((near[k] ≠ 0) and (cost[k, near[k]] > cost[k, j]))
25                  then near[k] := j;
26      }
27      return mincost;
28  }
```

- Graph represented as adjacency matrix **cost(n×n).**
- Minimum spanning tree constructed is stored in an array **t(n-1×2)**
- The cost of minimum spanning tree is returned as **mincost.**

Performance

- Prim's algorithm can be implemented efficiently using Binary Heap H:
- First, insert all edges adjacent to u into H
- At each step, extract the cheapest edge
- If an end-point, say v, is not in MST, include this edge and v to MST
- Insert all edges adjacent to v into H
- At most O(E) Insert/Extract-Min
- □ Total Time: O(E log E) = O(E log V)

Performance(speedup)

- In fact, Prim's algorithm can be sped up using a Fibonacci Heap F
- Instead of keeping edges in the heap, we keep distinct vertices. This avoids Θ(E) Extract-Min in the worst case
- At the beginning, each vertex (except source) is inserted into the heap, with key = ∞
- key represents distance between u and the vertex

Performance(speedup)

- Next, we scan all adjacent edges in u and update the distance of the corresponding vertices (using Decrease-Key)
- the vertex with the smallest key must be joined to u with the cheapest edge (since key = distance from u)
- So, we extract the minimum vertex, scan all its adjacent edges, and update corresponding vertices …

Performance(speedup)

- The process is repeated until all vertices in the heap are gone
- □ MST obtained !
- **Running Time**:
- O(V) Insert/Extract-Min
- •At most O(E) Decrease-Key
- □ Total Time: O(E + V log V)

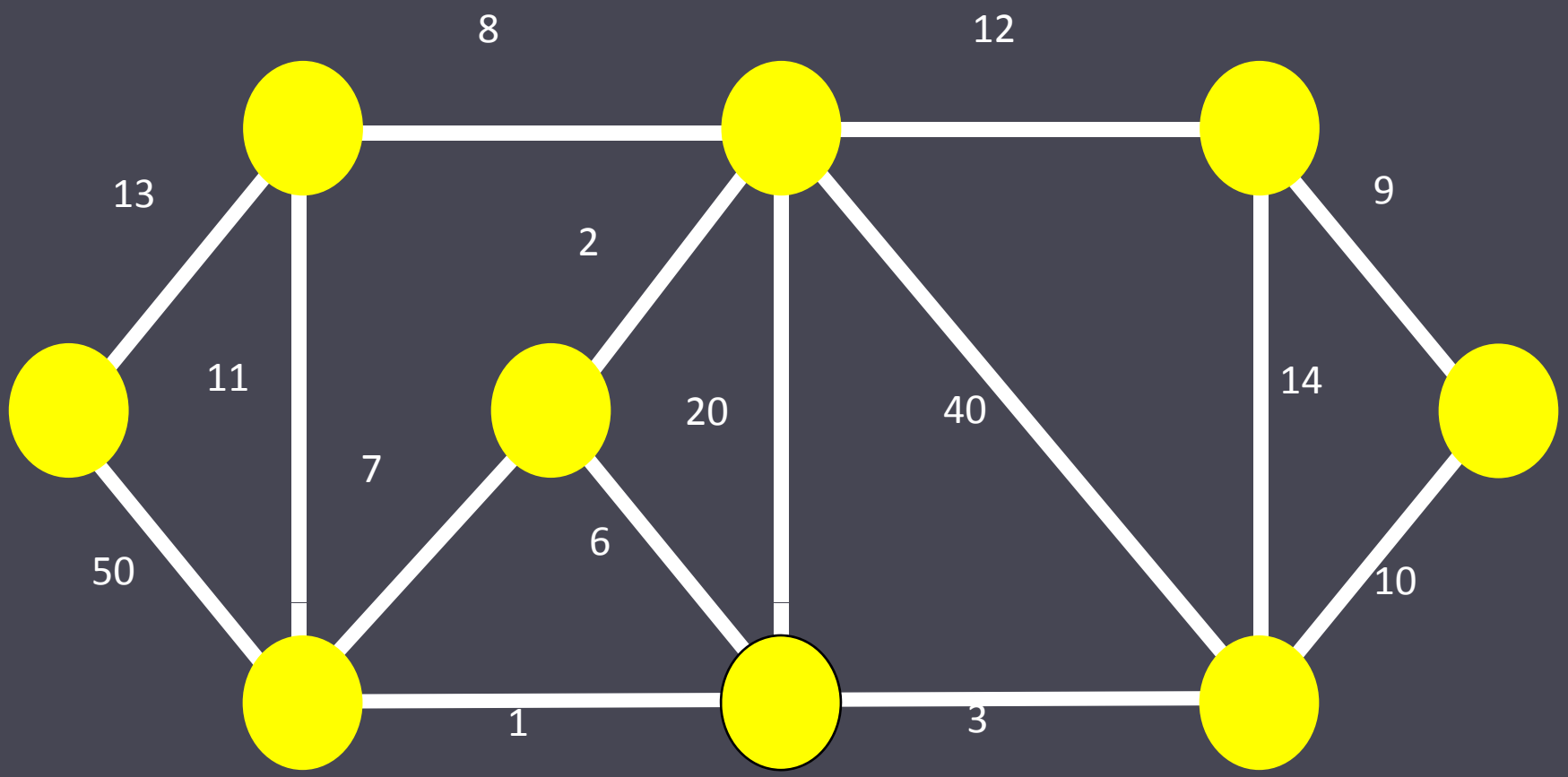
Prim with heaps:

1. Make a heap of values (vertex,edge,weight(edge))
2. initially (v,-,infinity) for each vertex
3. let tree T be empty
4. while (T has fewer than n vertices)
5. {
6. let (v,e,weight(e)) have the smallest weight in the heap
7. remove (v,e,weight(e)) from the heap
8. add v and e to T
9. for each edge f=(u,v)
10. if u is not already in T
11. find value (u,g,weight(g)) in heap
12. if weight(f) < weight(g)
13. replace (u,g,weight(g)) with (u,f,weight(f))
14. }

Analysis: Prim with heaps

- We perform n steps in which we remove the smallest element in the heap, and at most 2m steps in which we examine an edge f=(u,v).
- For each of those steps, we might replace a value on the heap, reducing it's weight. (You also have to find the right value on the heap, but that can be done easily enough by keeping a pointer from the vertices to the corresponding values.)
- I haven't described how to reduce the weight of an element of a binary heap, but it's easy to do in O(log n) time.
- Alternately by using a more complicated data structure known as a Fibonacci heap, you can reduce the weight of an element in constant time. The result is a total time bound of O(m + n log n).

Boruvka's Algorithm



Boruvka's Algorithm

- Otakar Borůvka
  - Inventor of MST
  - Czech scientist
  - Introduced the problem
  - The original paper was written in Czech in 1926.
  - The purpose was to efficiently provide electric coverage of Bohemia.

Boruvka's Algorithm

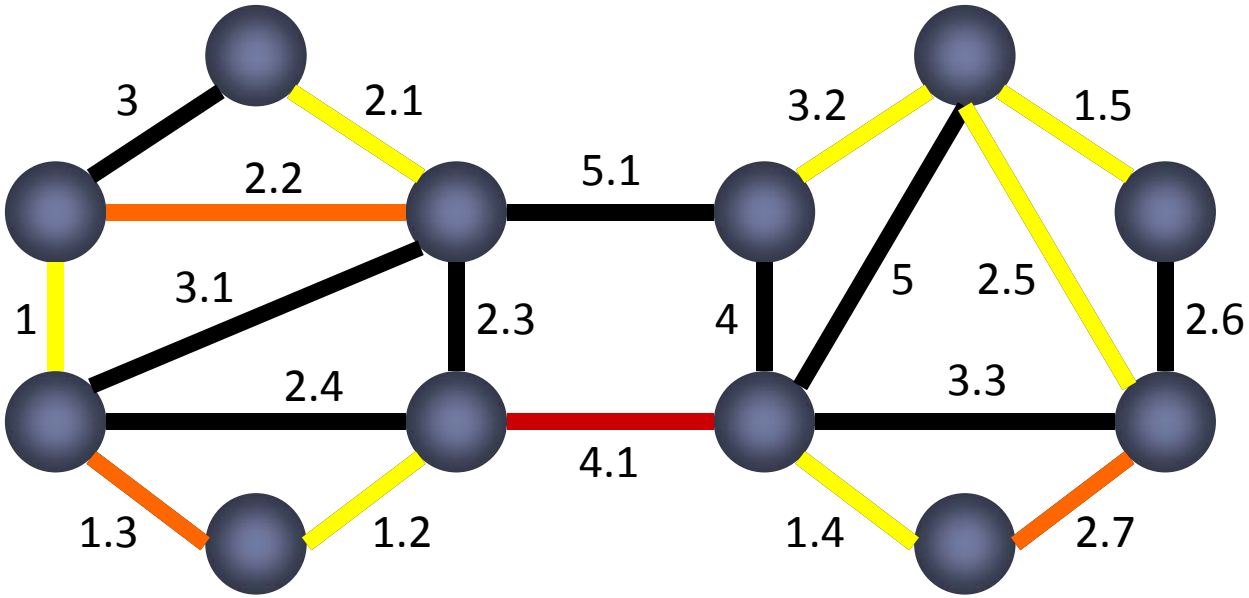
- Boruvka Approach:
  - Prim "in parallel"
  - Repeat the following procedure until the resulting graph becomes a single node.
    - For each node u, mark its lightest incident edge.
    - Now, the marked edges form a forest F. Add the edges of F into the set of edges to be reported.
    - Contract each maximal subtree of F into a single node.

Boruvka's algorithm

- The idea is to do steps like Prim's algorithm, in parallel all over the graph at the same time.
- Boruvka's algorithm:

1. make a list L of n trees, each a single vertex
2. while (L has more than one tree)
3. for each T in L, find the smallest edge connecting T to G-T
4. add all those edges to the MST
5. (causing pairs of trees in L to merge)

Boruvka's Algorithm - Example



Analysis: Boruvka's algorithm

- This is similar to merge sort. Each pass reduces the number of trees by a factor of two, so there are O(log n) passes.
- Each pass takes time O(m) (first figure out which tree each vertex is in, then for each edge test whether it connects two trees and is better than the ones seen before for the trees on either endpoint) so the total is O(m log n).

Usage of Minimum Spanning Trees

- Network design:
  - telephone, electrical, hydraulic, TV cable, computer, road
- Approximation algorithms for NP-hard (non-deterministic polynomial-time hard) problems:
  - traveling salesperson problem
- Cluster Analysis

Implementation Issues: 1

- How is the graph implemented?
  - Assume that we just added node u to the tree.
  - The distance of the nodes adjacent to u to the tree may now be decreased.
  - There must be fast access to all the adjacent vertices.
  - So using **adjacency lists** seems better
- How should the set of non-tree vertices be represented?
  - The operations are:
    - build set
    - delete node closest to tree
    - decrease the distance of a non-tree node from the tree
    - check whether a node is a non- tree node

Implementation Issues: 2

- How should the set of non-tree vertices be represented?
  - A priority queue PQ may be used with the priority D[v] equal to the minimum *distance* of each non-tree vertex v to the tree.
  - Each item in PQ contains: D[v], the vertex v, and the shortest distance edge (v, u) where u is a tree node
- This means:
  - build a PQ of non-tree nodes with initial values -
    - fast build heap O (V)
    - building an unsorted list O(V)
    - building a sorted list O(V) (special case)

Implementation Issues:3

- delete node closest to tree (extractMin)
  - **O(lg *V*) if heap and**
  - **O(V) if unsorted list**
  - **O(1) sorted list**

- decrease the distance of a non-tree node to the tree
- We need to find the location i of node v in the priority queue and then execute (decreasePriorityValue(i, p)) where p is the new priority
- decreasePriorityValue(i, p)
  - O(lg V) for heap,
  - O(1) for unsorted list
  - O(***V***) for sorted list (too slow)

Implementation Issues:4

- What is the location i of node v in a priority queue?
  - Find in Heaps, and sorted lists O(n)
  - Unsorted – if the nodes are numbered 1 to n and we use an array where node v is the v item in the array O(1)

Extended heap

- We will use extended heaps that contain a “handle” to the location of each node in the heap.
- When a node is not in PQ the “handle” will indicate that this is the case
- This means that we can access a node in the extended heap in O(1), and check v $\in$ PQ in O(1)
- Note that the “handle” must be updated whenever a heap operation is applied

Implementation Issues: 5

2. Unsorted list

- Array implementation where node v can be accesses as PQ[v] in O(1), and the value of PQ[v] indicates when the node is not in PQ.

Notes about Kruskal's algorithm

- Algorithm looks easier than Prim's but is harder to implement (checking for cycles!)
- Cycle checking: a cycle is created iff added edge connects vertices in the same connected component
- *Union-find* algorithms

Applications

Minimum spanning trees are useful in constructing networks, by describing the way to connect a set of sites using the smallest total amount of wire.

Cable TV

- One example is a cable TV company laying cable to a new neighborhood.
- If it is constrained to bury the cable only along certain paths, then there would be a graph representing which points are connected by those paths.
- Some of those paths might be more expensive, because they are longer, or require the cable to be buried deeper.
- A spanning tree for that graph would be a subset of those paths that has no cycles but still connects to every house.
- There might be several spanning trees possible.
- A minimum spanning tree would be one with the lowest total cost.

Circuit design

- In the design of electronic circuitry, it is often necessary to wire some pins together in order to make them electrically equivalent.
- A minimum spanning tree needs the least amount of wire to interconnect a set of points.

Islands connection

- Suppose we have a group of islands that we wish to link with bridges so that it is possible to travel from one island to any other in the group.
- Further suppose that the government wishes to spend the minimum amount on this project.
- The engineers are able to calculate a cost for a bridge linking each possible pair of islands.
- The set of bridges that will enable one to travel from any island to any other at the minimum cost to the government is the minimum spanning tree.

Clustering gene expression data

- Minimum spanning trees also provide a reasonable way for clustering points in space into natural groups.
- For example,
- Y. Xu, V. Olman, and D. Xu. Clustering gene expression data using a graph-theoretic approach: An application of minimum spanning trees. Bioinformatics, 18:536–545, 2002.
- Ying Xu and his coworkers describe a new framework for representing a set of multi-dimensional gene expression data as a minimum spanning tree.
- A key property of this representation is that each cluster of the gene expression data corresponds to one subtree of the MST, which rigorously converts a multi-dimensional clustering problem to a tree partitioning problem.
- They have demonstrated that, although the inter-data relationship is greatly simplified in the MST representation, no essential information is lost for the purpose of clustering.

Clustering gene expression data

- They observe that there are two key advantages in representing a set of multi-dimensional data as an MST
- One is that the simple structure of a tree facilitates efficient implementations of rigorous clustering algorithms, which otherwise are highly computationally challenging.
- The other is that it can overcome many of the problems faced by classical clustering algorithms since an MST based clustering does not depend on detailed geometric shape of a cluster.
- A new software tool called EXCAVATOR, which stands for "EXpression data Clustering Analysis and VisualizATiOn Resource," has been developed based on this new framework.
- The clustering results on the gene expression data (1) from yeast Saccharomyces cerevisiae, (2) in response of human fibroblasts to serum, and (3) of Arabidopsis in response to chitin elicitation are very promising.

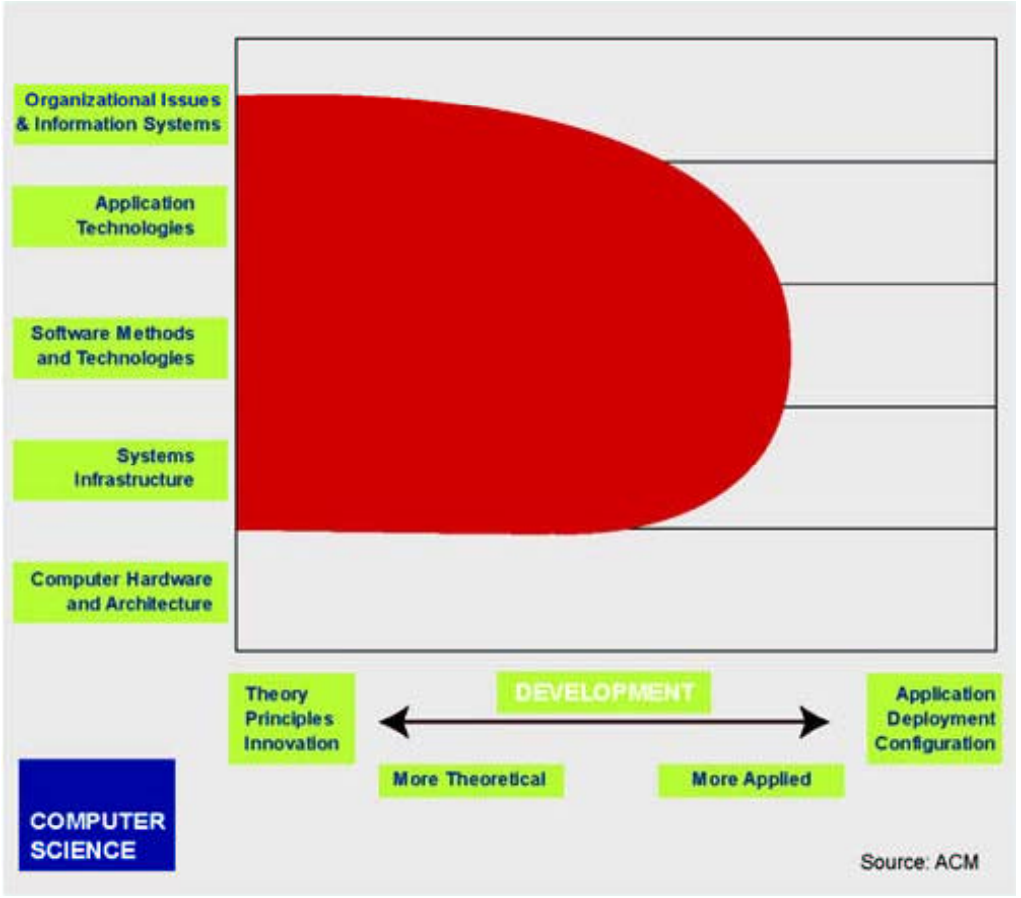
MST-based approximations

- In the traveling salesperson problem (TSP), we are given a complete undirected graph G that has weight function w associated with each edge, and we wish to find a tour of G with minimum weight.
- This problem has been shown to be NP-hard even when the weight function satisfies the triangle inequality, i.e., for all three vertices x, y, z ∈ V , w(x, z) ≤ w(x, y) + w(y, z).
- The triangle inequality arises in many practical situations.
- It can be shown that the following strategy delivers an approximation algorithm with a ratio bound of 2 for the traveling salesperson problem with triangle inequality.

MST-based approximations

- First, find a minimum spanning tree T for the given graph.
- Then double the MST and construct a tour T 0.
- Finally, add shortcuts so that no vertex is visited more than once, which is done by a preorder tree walk. The resulting tour is of length no more than twice of the optimal. It can also be shown that an MST-based approach also provides a good approximation for the Steiner tree problems.

Thank you


Organizational Issues & Information Systems
Application Technologies
Software Methods and Technologies
Systems Infrastructure
Computer Hardware and Architecture
Theory Principles Innovation
DEVELOPMENT
Application Deployment Configuration
More Theoretical
More Applied
COMPUTER SCIENCE
Source: ACM

- In Kruskal's method for finding a minimum spanning tree, how does the algorithm know when the addition of an edge will generate a cycle?
- Kruskal's algorithm for finding a minimum spanning tree employs a minheap to store the candidate edges. If a graph has *m* edges, explain why the time complexity for this algorithm must be **O(m logm).**