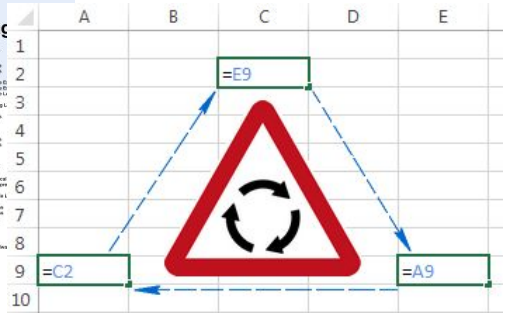
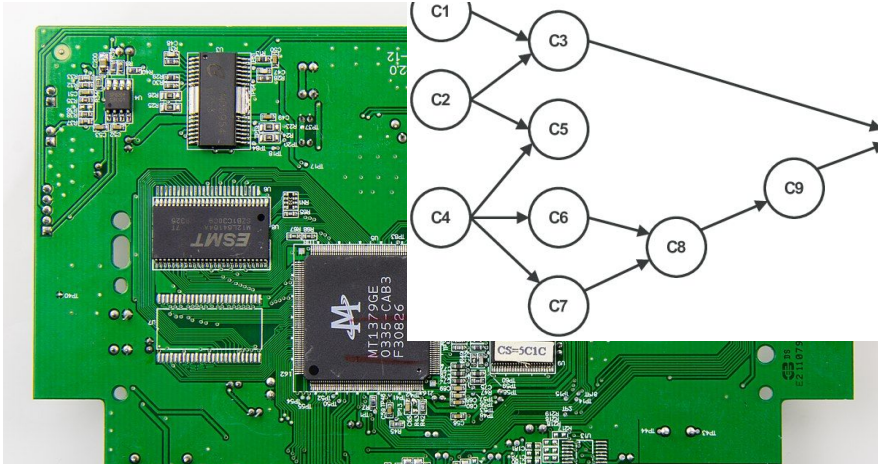
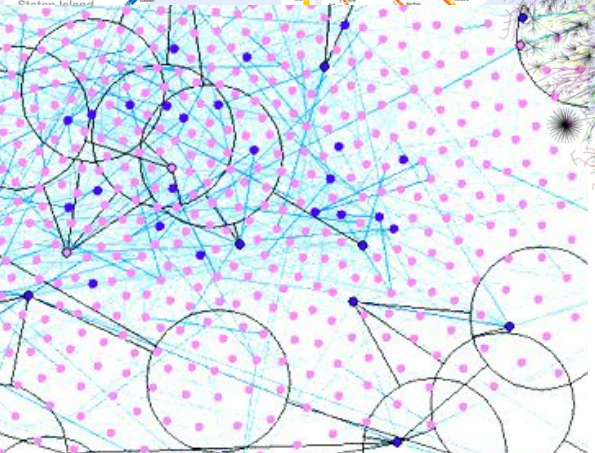
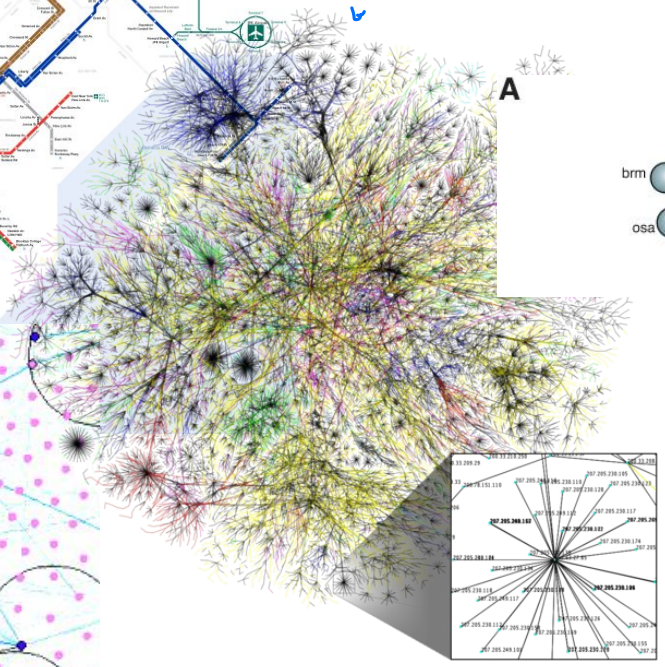
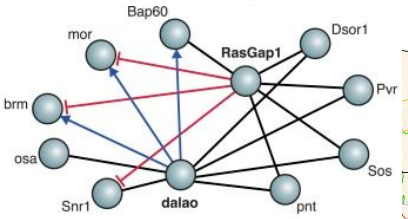

New York City Subway Diag
The Bronx
Queens
Manhattan
Brooklyn


A B C D E
1 2 3 4 5 6 7 8 9 10
=E9
=C2
=A9


C1
C2
C3
C4
C5
C6
C7
C8
C9
MT1379GE
0335-CAB3
F3082G
CS-5C1C

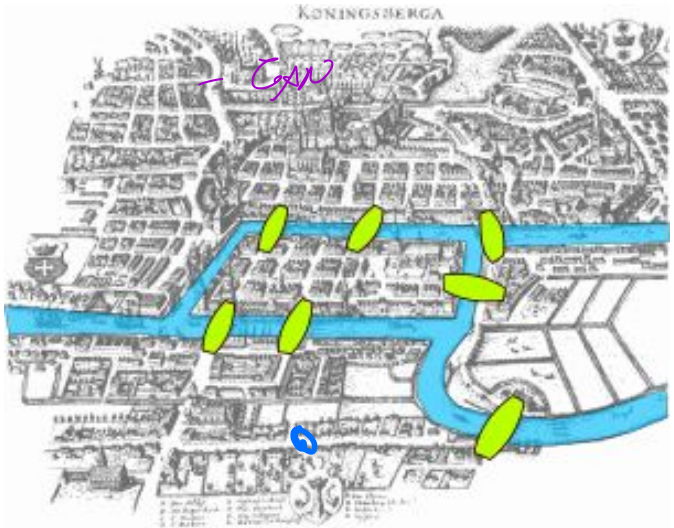


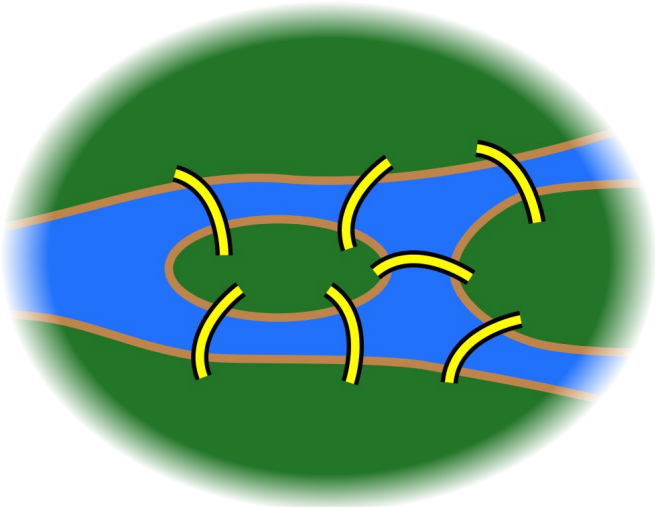

A

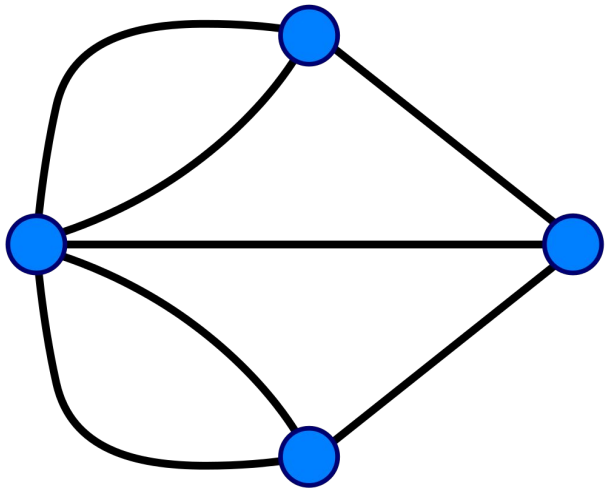

Bap60
mor
Dsor1
RasGap1
Pvr
brm
osa
Sos
Snr1
dalao
pnt

United States
transmission grid
Source: FEMA
kV
115
138
161
230
345
500

Euler's "Seven Bridges of Königsberg (1736)



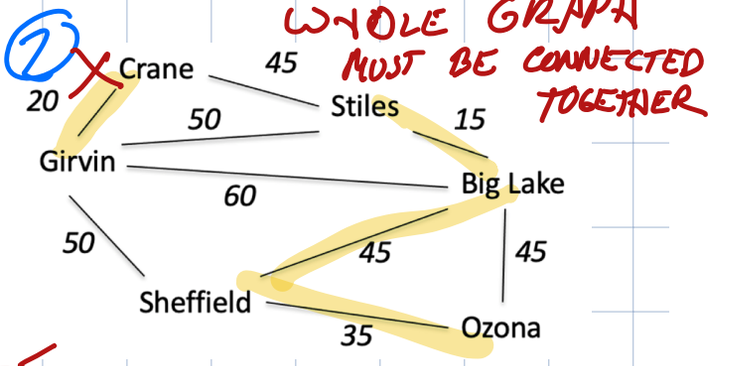


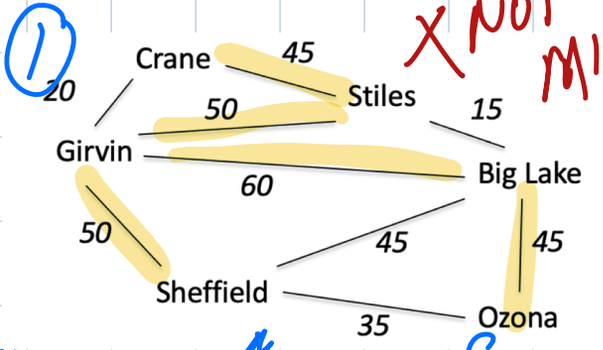


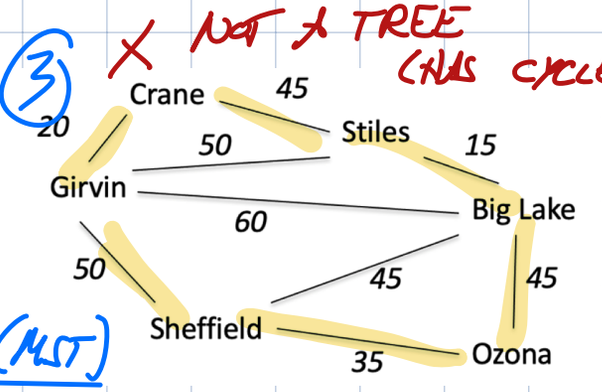
Problem discussed in lecture: constructing an electrical network in Moravia (Boruvka, 1926)

Creating electrical networks

Requirements
- Use smallest length of wire possible
  => Minimum cost ①
- Bring electricity to all cities
  => Spanning all cities ②
- Don't want unnecessary connections between cities
  => No cycles ⇒ MUST BE A TREE ③

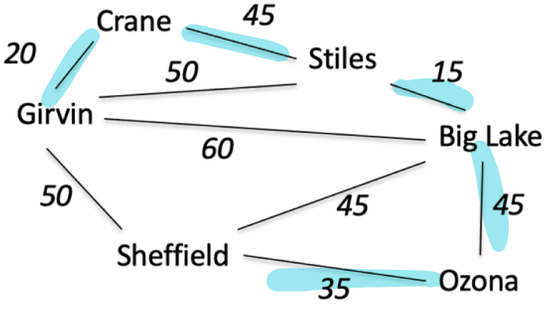


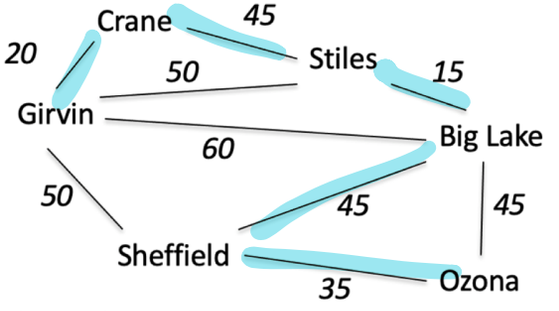




GOAL: FIND A MINIMUM SPANNING TREE (MST)

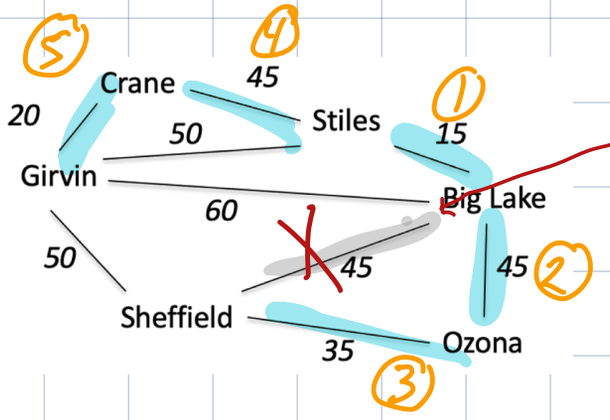
This graph has two possible MSTs:





- TREE ✓
- MIN COST ✓
- SPANNING ✓

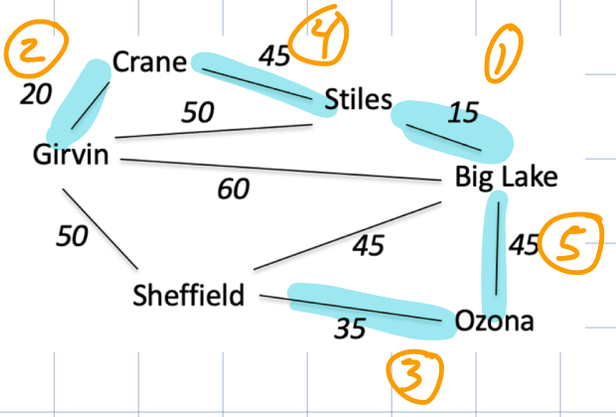
Examples of MST Algorithms



Jarnik's (1930) / Prim's (1956)

- Choose random starting node, build tree from there
- Of all possible edges coming out of tree, add smallest edge that doesn't introduce a cycle
  => Minimum, tree
- Stop once you've connected all cities

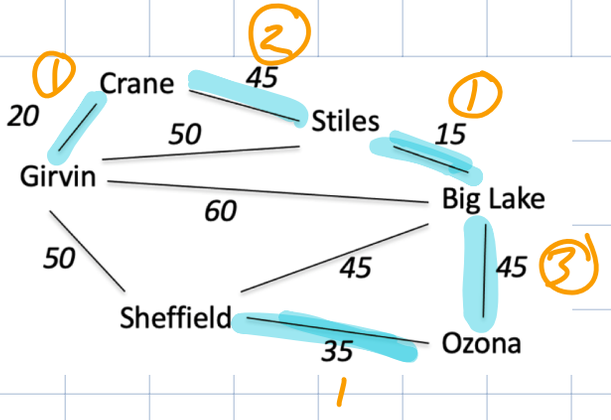
=> Recompute best edge each time we add to tree



Kruskal's

- First, sort all edges by weight (ie, make a priority queue of edges)
- For each step, select smallest edge that doesn't introduce a cycle
- Stop when you have all cities in a single tree

=> Don't need to recompute PQ each time, but has one big data structure



Sollin/Boruvka's

- Cities pair off to form "optimal" collectives
- Collectives repeatedly try to connect to each other until we have a spanning tree

=> Can work in parallel

	Jarnik/Prim	Kruskal	Boruvkas/Sollin
Approach	pick random start node each iteration, add the cheapest edge that won't create a cycle	sort the edges from cheapest to most expensive; each iteration adds cheapest edge that won't make a cycke	do kruskal's algorithm in parallel (each node be a component, keep merging components into an overall tree)
Data Structure Used	**priority queue** of edges that connect node "in" the tree to one outside the tree	**sorted list** of edges	sorted list of edges **connected to each component**
Must Be Able To	**maintain the priority queue** as nodes/edges were added to the tree, check that next cheapest edge **won't create a cycle**	check that next cheapest edge **won't create a cycle**	work and coordinate process in **parallel**

⇒ TRADEOFFS IN COMPUTATION + DATA STRUCTURES