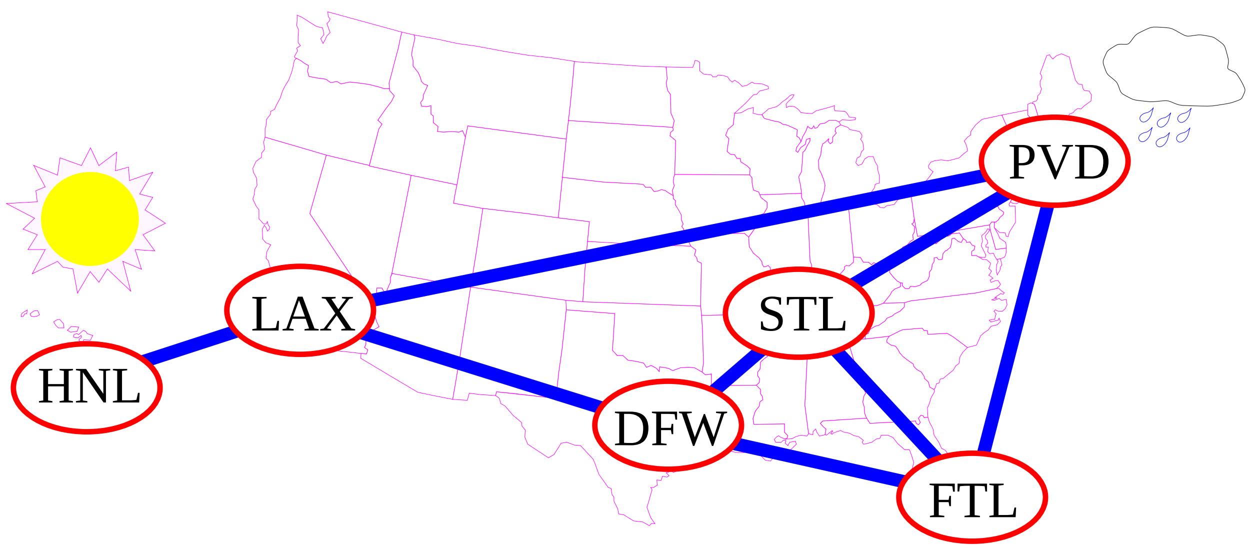
GRAPHS

- Definitions
- Examples
- The Graph ADT



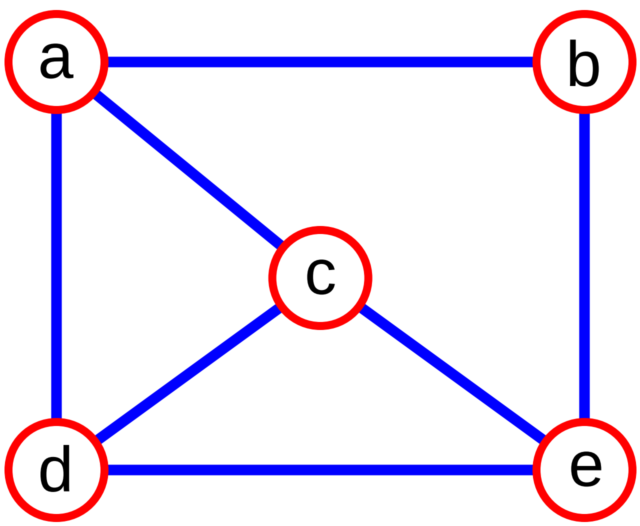
What is a Graph?

- A **graph G** = (**V**,**E**) is composed of:

  V: set of *vertices*

  E: set of *edges* connecting the *vertices* in **V**

- An **edge e** = (u,v) is a pair of vertices
- Example:



V= {a,b,c,d,e}

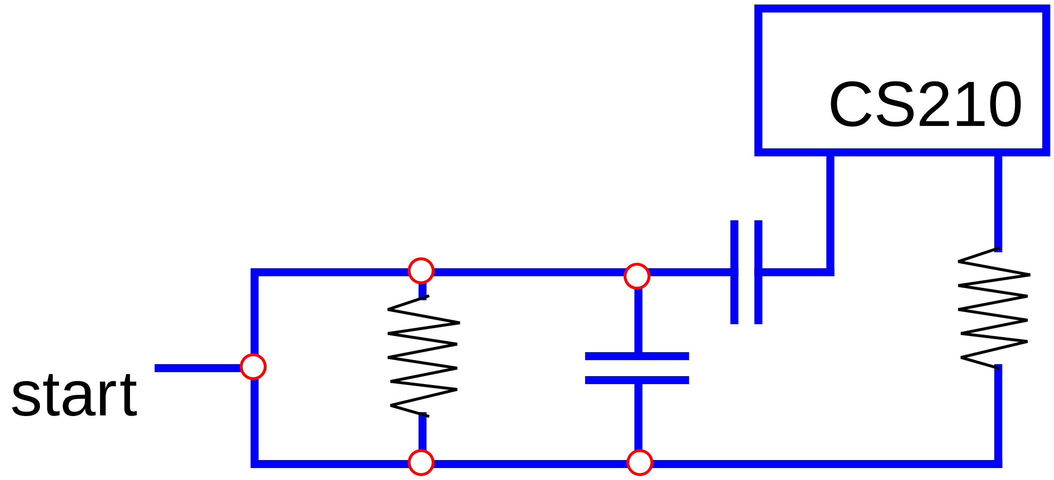
E= {(a,b),(a,c),(a,d),
(b,e),(c,d),(c,e),
(d,e)}

Edge Types

- Directed edge
  - ordered pair of vertices (u,v)
  - first vertex u is the origin
  - second vertex v is the destination
  - e.g., a flight
- Undirected edge
  - unordered pair of vertices (u,v)
  - e.g., a flight route
- Directed graph
  - all the edges are directed
  - e.g., route network
- Undirected graph
  - all the edges are undirected
  - e.g., flight network

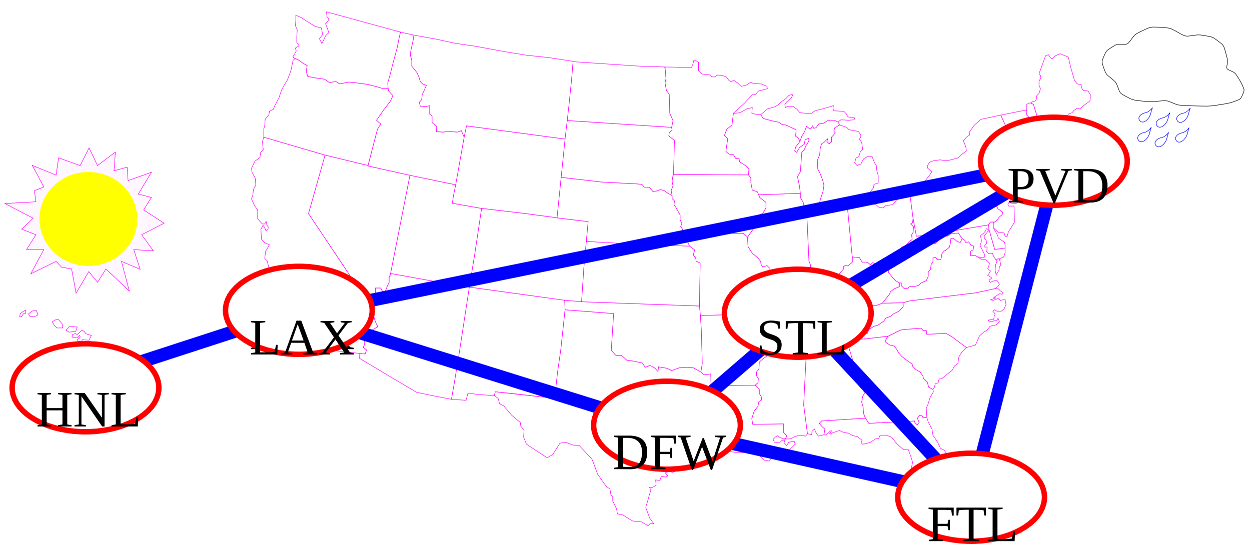
Applications

- electronic circuits

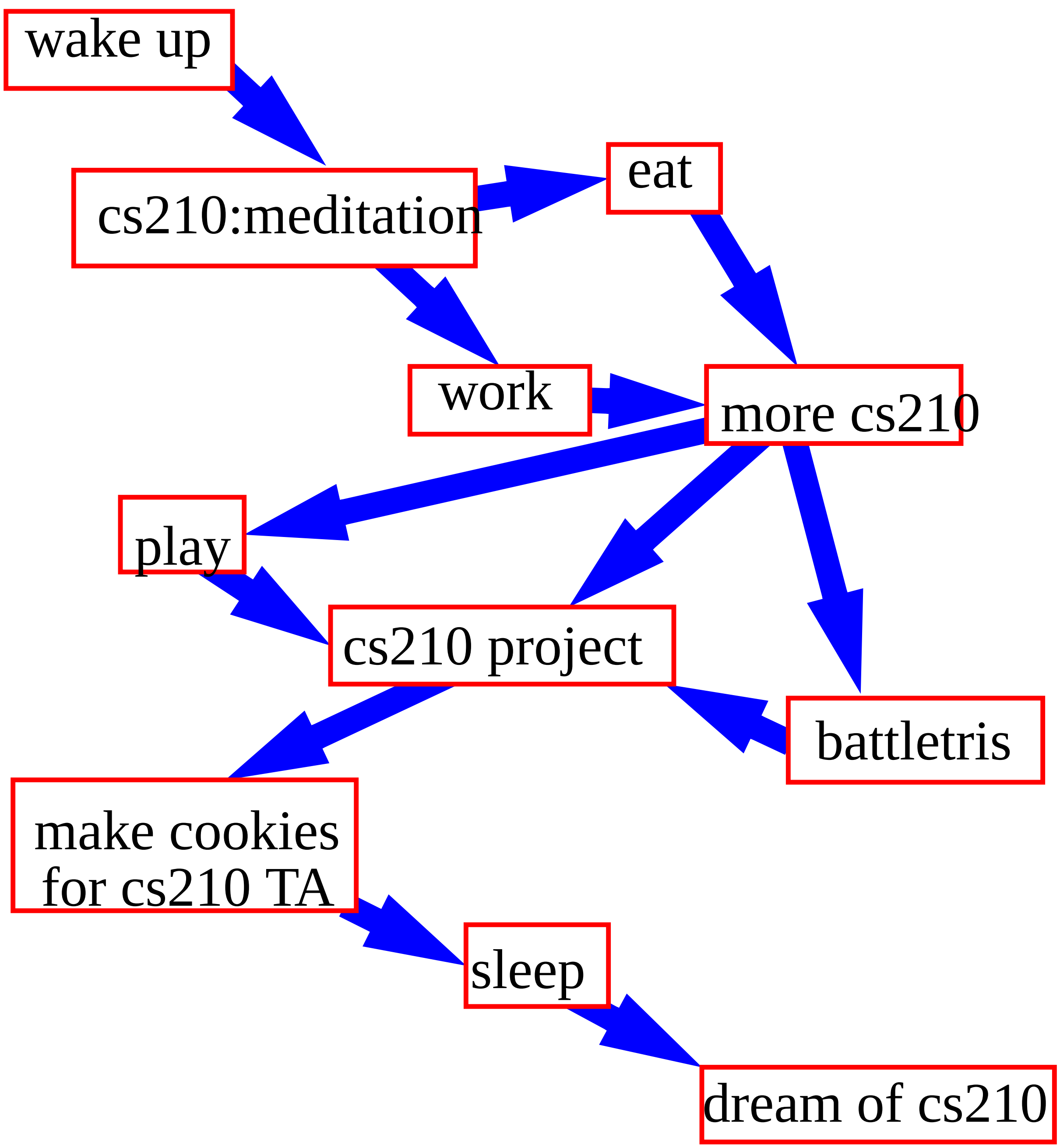


find the path of least resistance to CS210

- networks (roads, flights, communications)

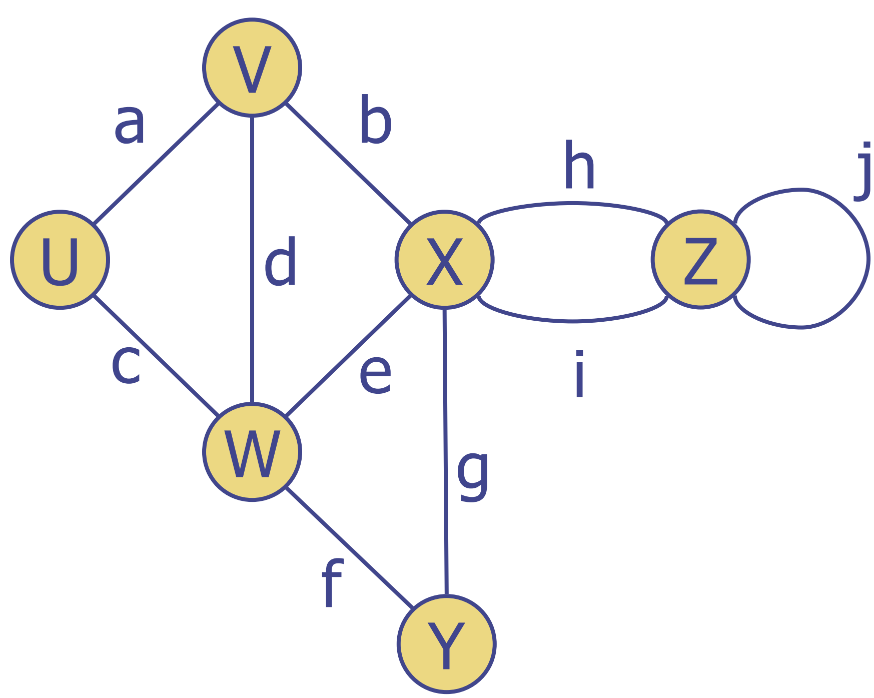


A typical student day



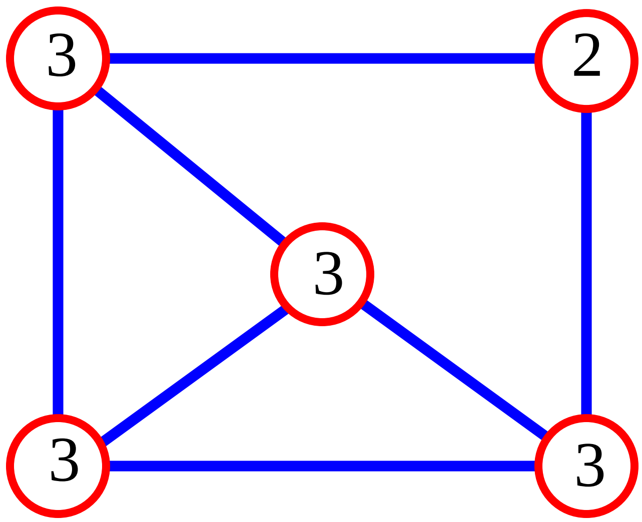
Terminology

- End vertices (or endpoints) of an edge
  - U and V are the endpoints of a
- Edges incident on a vertex
  - a, d, and b are incident on V
- Adjacent vertices
  - U and V are adjacent
- Degree of a vertex
  - X has degree 5
- Parallel edges
  - h and i are parallel edges
- Self-loop
  - j is a self-loop



Graph Terminology

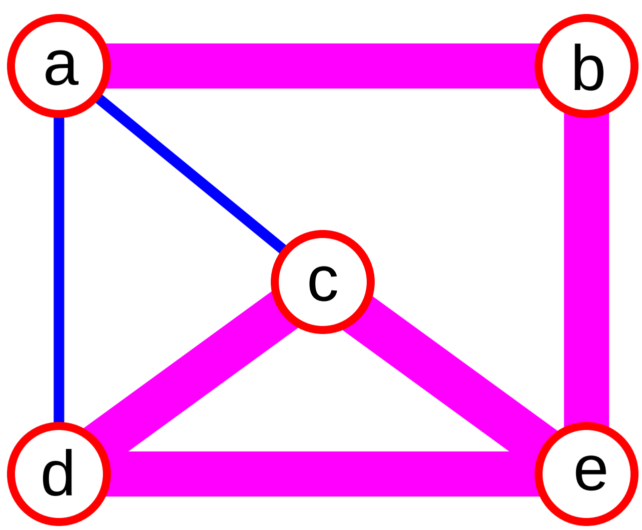
- adjacent vertices: connected by an edge
- degree (of a vertex): # of adjacent vertices



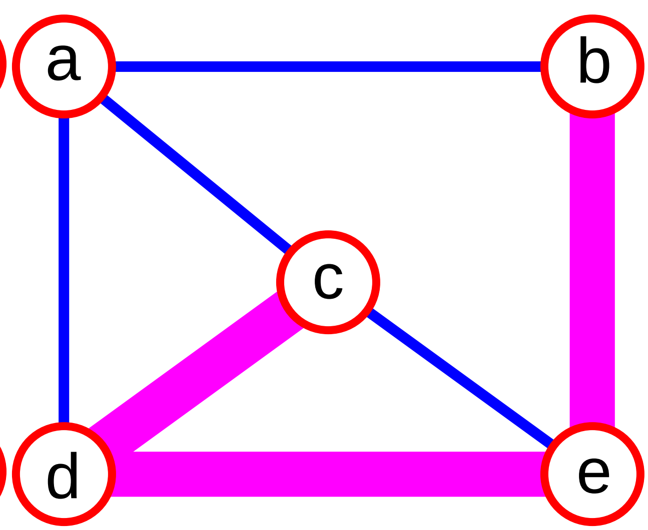
$$\sum_{v \in V} \deg(v) = 2(\#\ \text{edges})$$

- Since adjacent vertices each count the adjoining edge, it will be counted twice

path: sequence of vertices $v_1, v_2, \ldots v_k$ such that consecutive vertices v_i and v_{i+1} are adjacent.



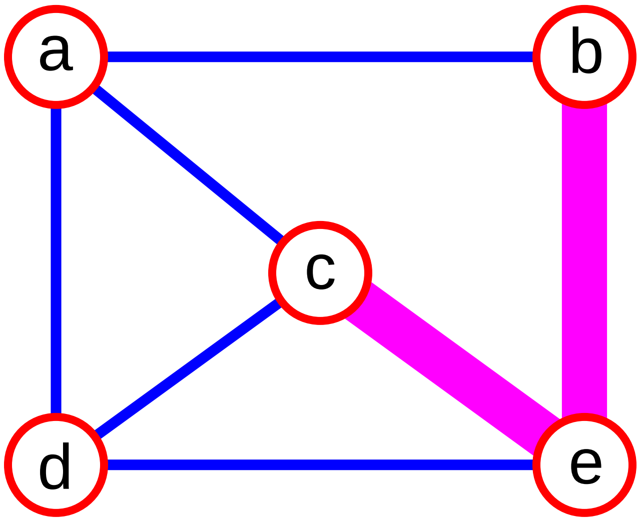
a b e d c



b e d c

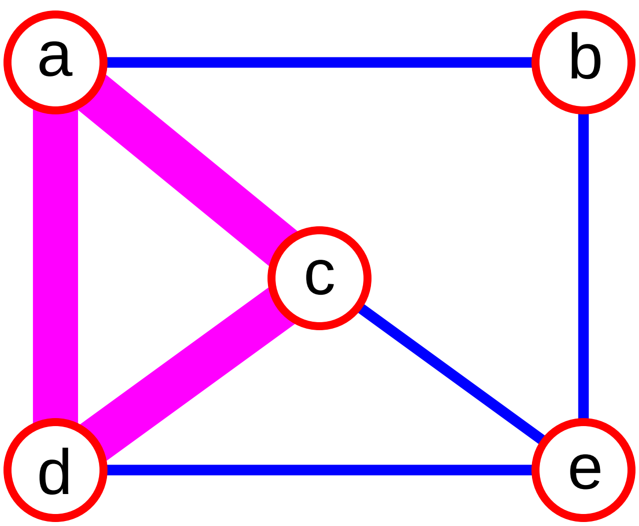
More Graph Terminology

- simple path: no repeated vertices

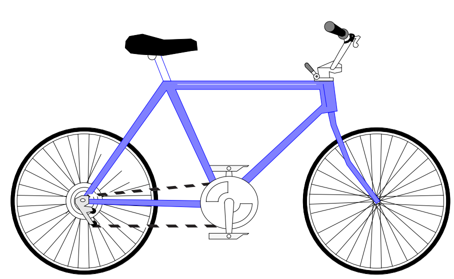


b e c

- cycle: simple path, except that the last vertex is the same as the first vertex

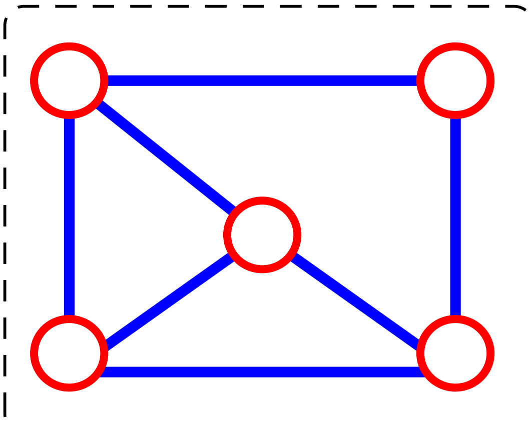


a c d a

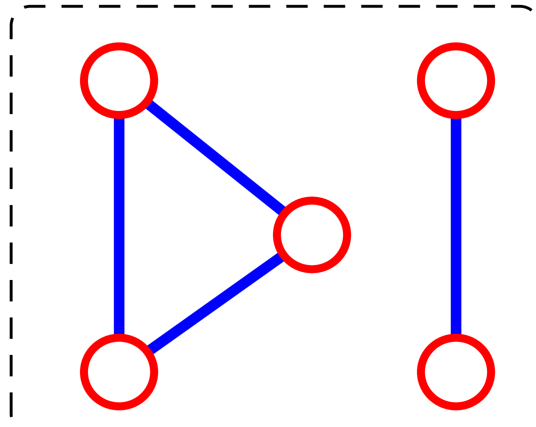


Even More Terminology

- connected graph: any two vertices are connected by some path

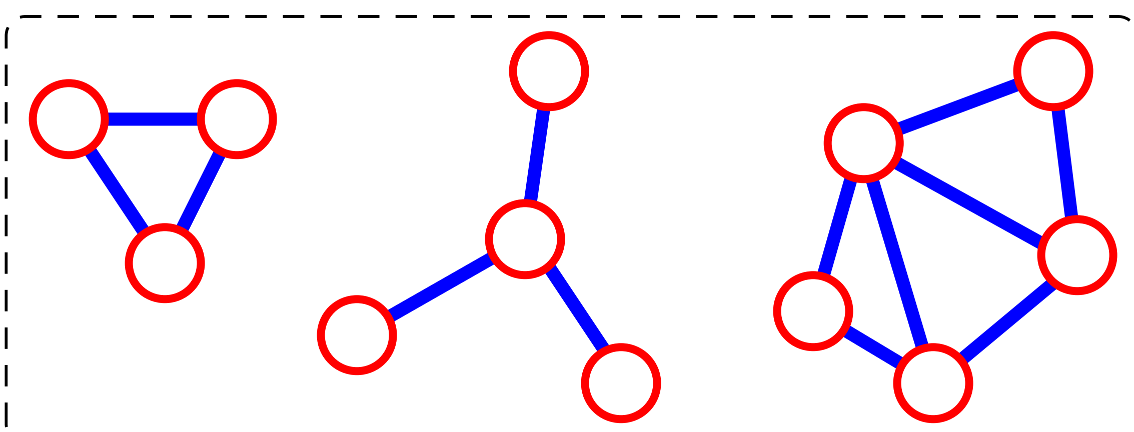


connected



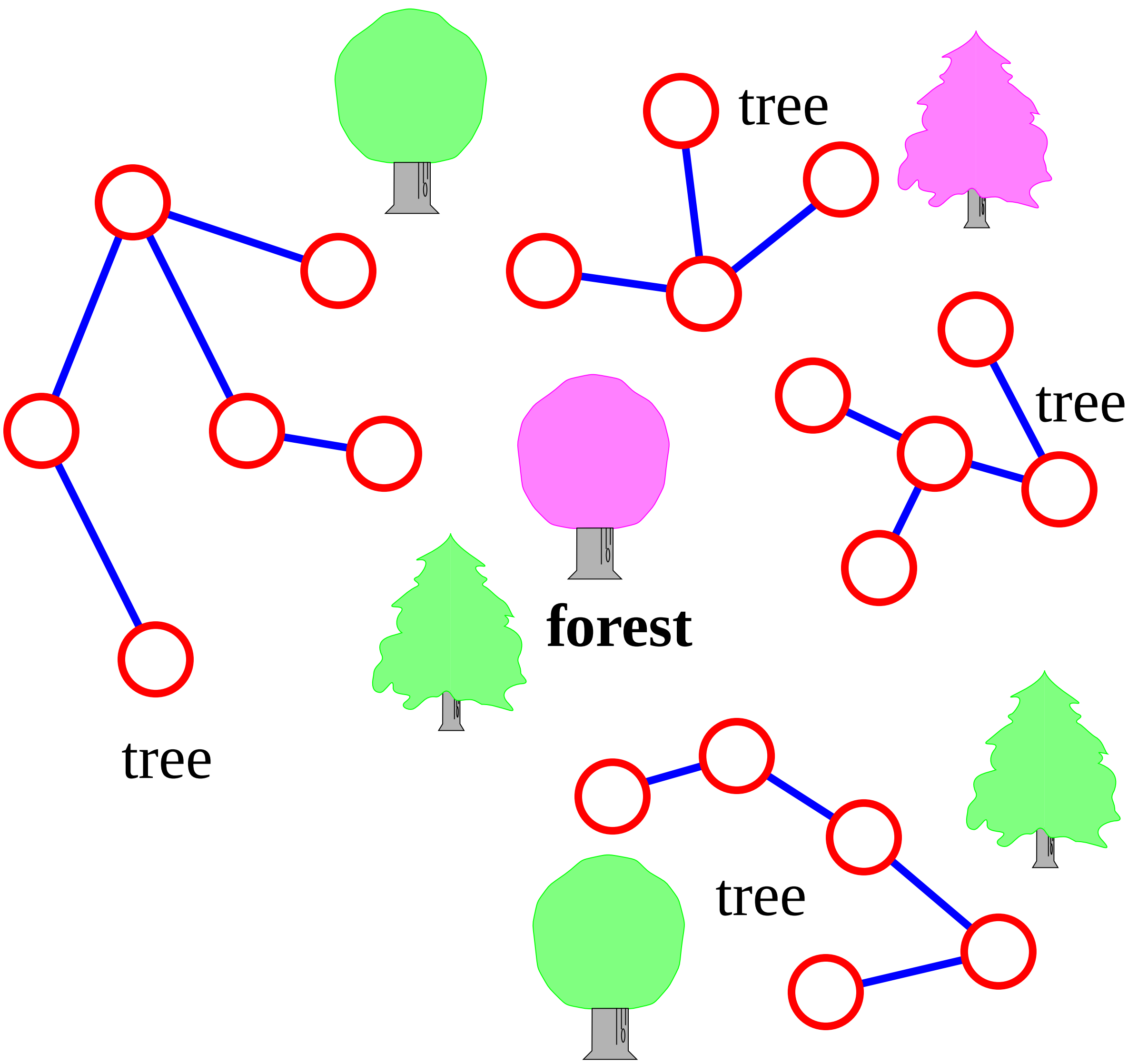
not connected

- subgraph: subset of vertices and edges forming a graph
- connected component: maximal connected subgraph. E.g., the graph below has 3 connected components.



Another Terminology Slide!

- (free) tree - connected graph without cycles
- forest - collection of trees



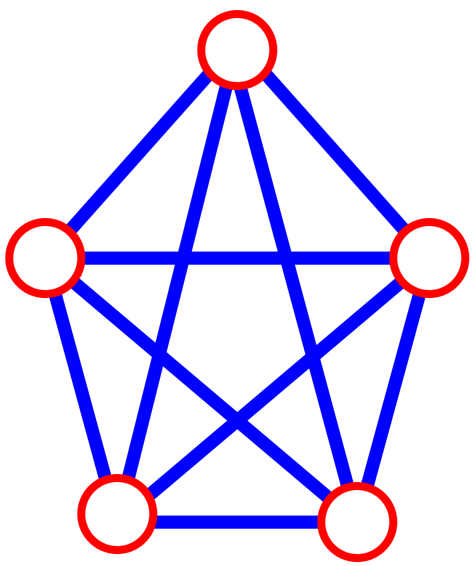
Connectivity

Let **n** = #vertices
m = #edges

- complete graph - all pairs of vertices are adjacent

$$\mathbf{m} = (1/2)\sum_{v \in \mathbf{V}} \deg(\mathbf{v}) = (1/2)\sum_{v \in V} (\mathbf{n} - 1) = \mathbf{n}(\mathbf{n}-1)/2$$

- Each of the **n** vertices is incident to **n** - 1 edges, however, we would have counted each edge twice Therefore, intuitively, **m** = **n**(**n**-1)/2.



n = 5
m = (5 ∗ 4)/2 = 10

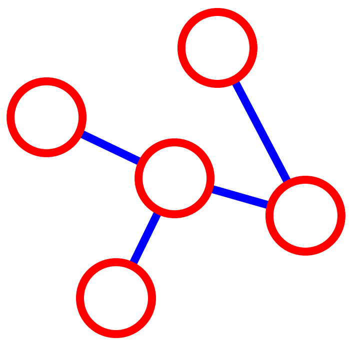
- Therefore, if a graph is ***not*** complete,

$$\mathbf{m} < \mathbf{n}(\mathbf{n}-1)/2$$

More Connectivity

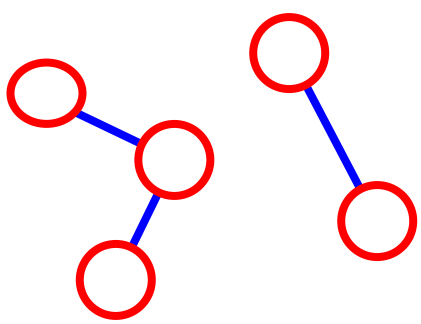
n = #vertices
m = #edges

- For a tree **m** = **n** - 1



n = 5
m = 4

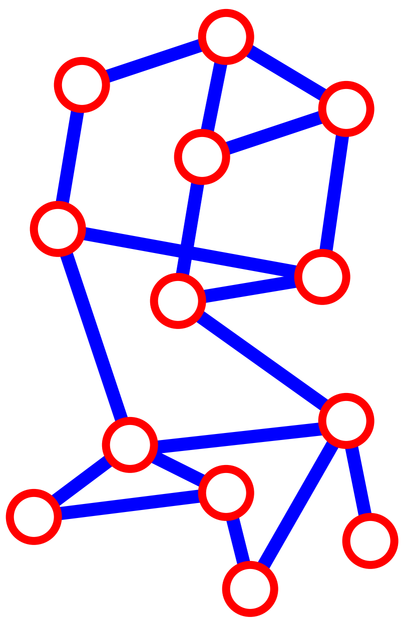
- If **m** < **n** - 1, G is not connected



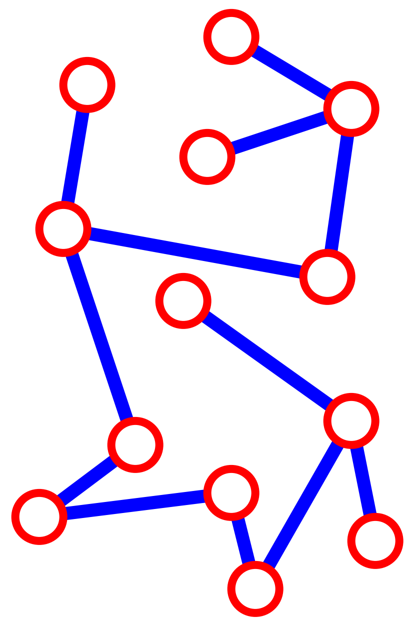
n = 5
m = 3

Spanning Tree

- A **spanning tree** of **G** is a subgraph which
  - is a tree
  - contains all vertices of **G**



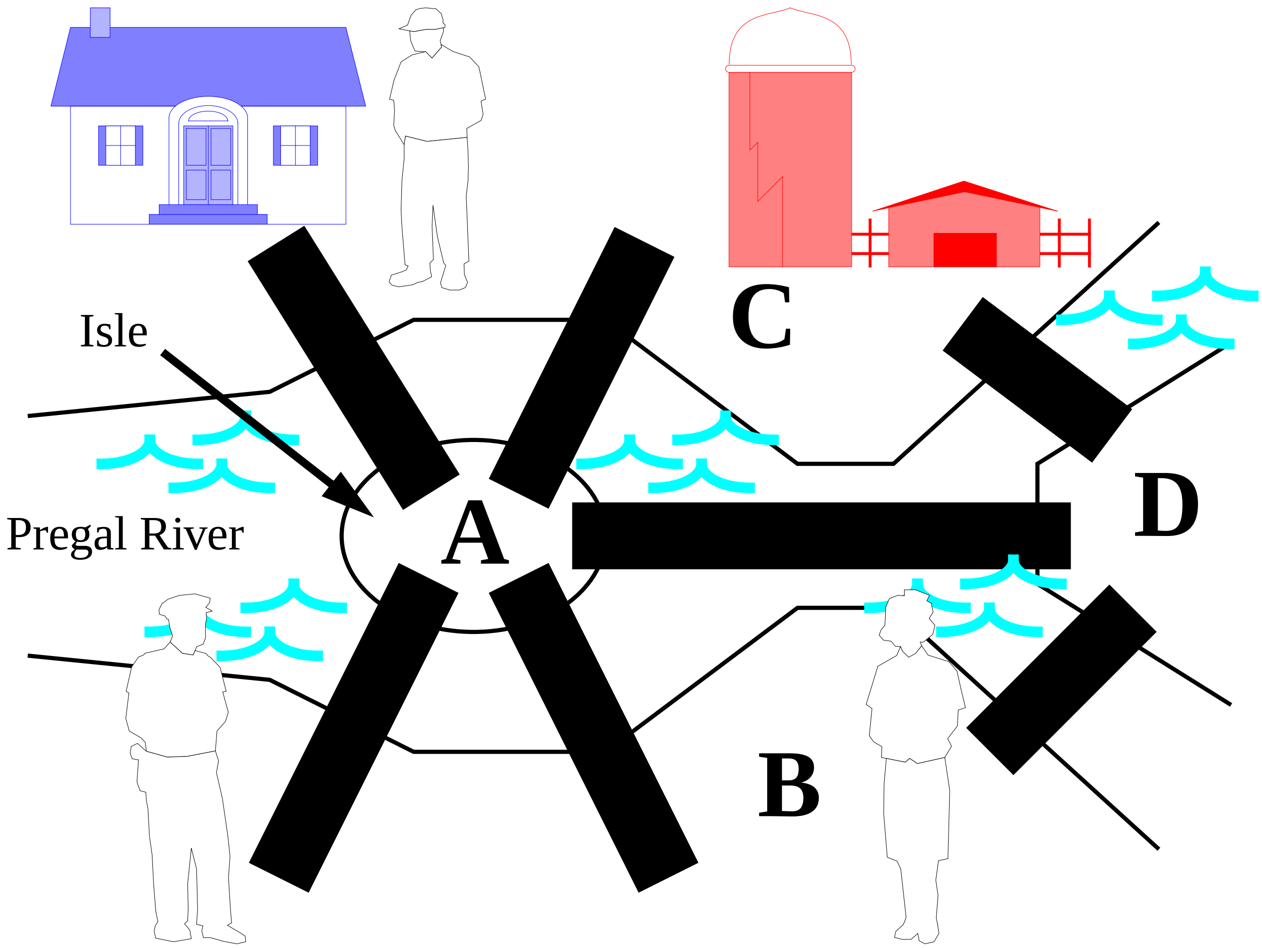
G



spanning tree of **G**

- Failure on any edge disconnects system (least fault tolerant)

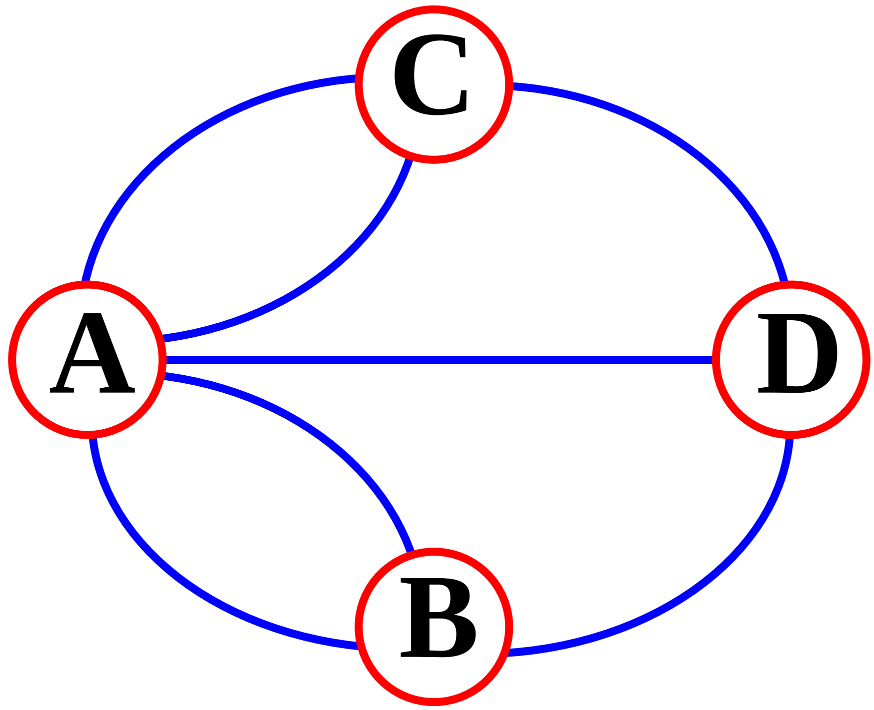
Euler and the Bridges of Koenigsberg



Can one walk across each bridge exactly once and return at the starting point?

- Consider if you were a UPS driver, and you didn't want to retrace your steps.
- In 1736, Euler proved that this is not possible

Graph Model(with parallel edges)



- Eulerian Tour: path that traverses every edge exactly once and returns to the first vertex
- Euler's Theorem: A graph has a Eulerian Tour if and only if all vertices have even degree

The Graph ADT

- Vertices and edges
  - are positions
  - store elements
- Accessor methods
  - endVertices(e): an array of the two endvertices of e
  - opposite(v, e): the vertex opposite of v on e
  - areAdjacent(v, w): true iff v and w are adjacent
  - replace(v, x): replace element at vertex v with x
  - replace(e, x): replace element at edge e with x
- Update methods
  - insertVertex(o): insert a vertex storing element o
  - insertEdge(v, w, o): insert an edge (v,w) storing element o
  - removeVertex(v): remove vertex v (and its incident edges)
  - removeEdge(e): remove edge e
- Iterator methods
  - incidentEdges(v): edges incident to v
  - vertices(): all vertices in the graph
  - edges(): all edges in the graph