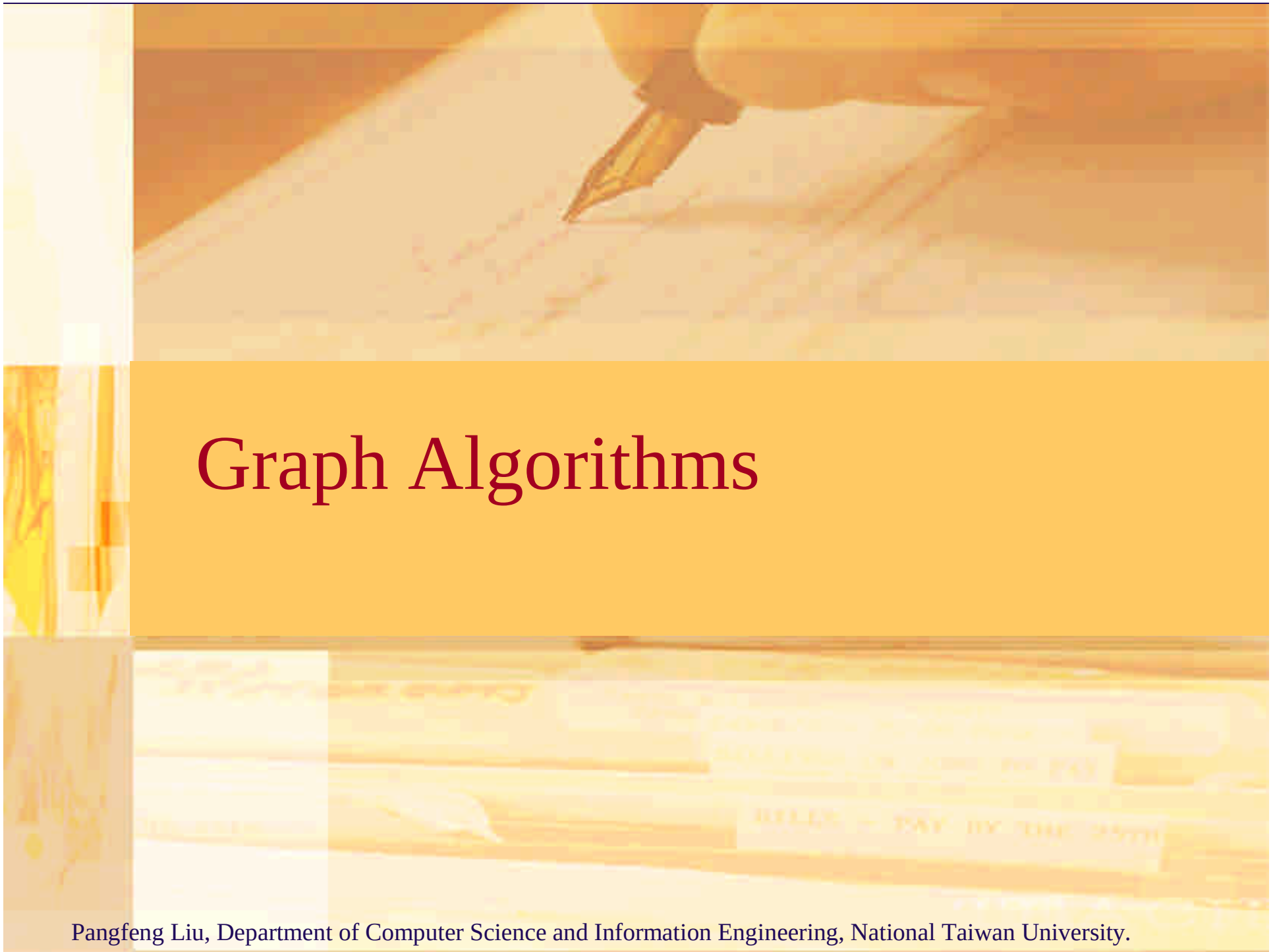


Graph Algorithms

Pangfeng Liu, Department of Computer Science and Information Engineering, National Taiwan University.

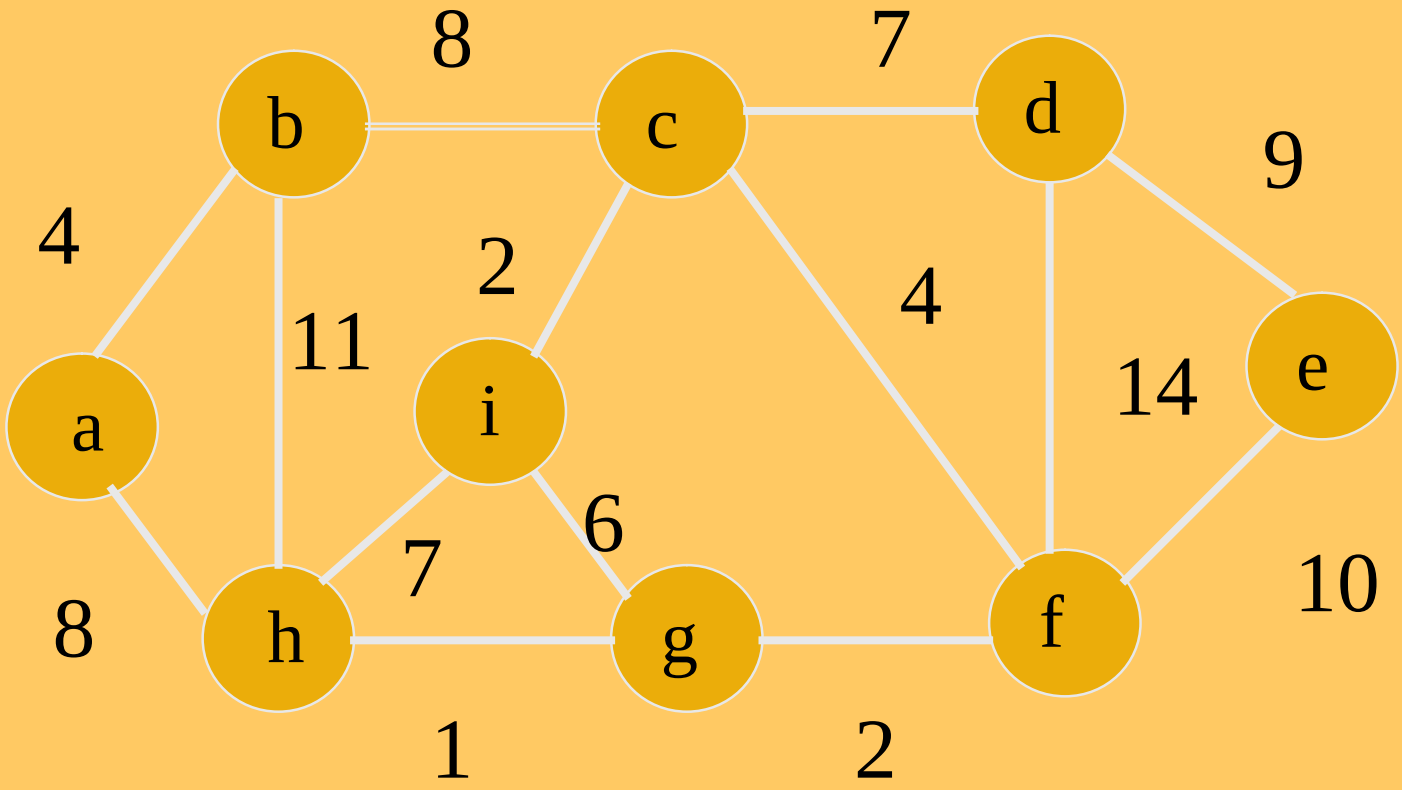
Minimum Spanning Tree

- A subset of edges that have the following properties.
  - It is a tree
    - Connected
    - No cycle
  - With the minimum total edge costs.

Prim's Algorithm

- Prim-MST(G)
  - Select an arbitrary node to start the tree from.
  - While there are still non-tree nodes
    - Select the edge of minimum weight between a tree and non-tree node
    - Add the selected edge and node to the tree.

Prim’s Algorithm



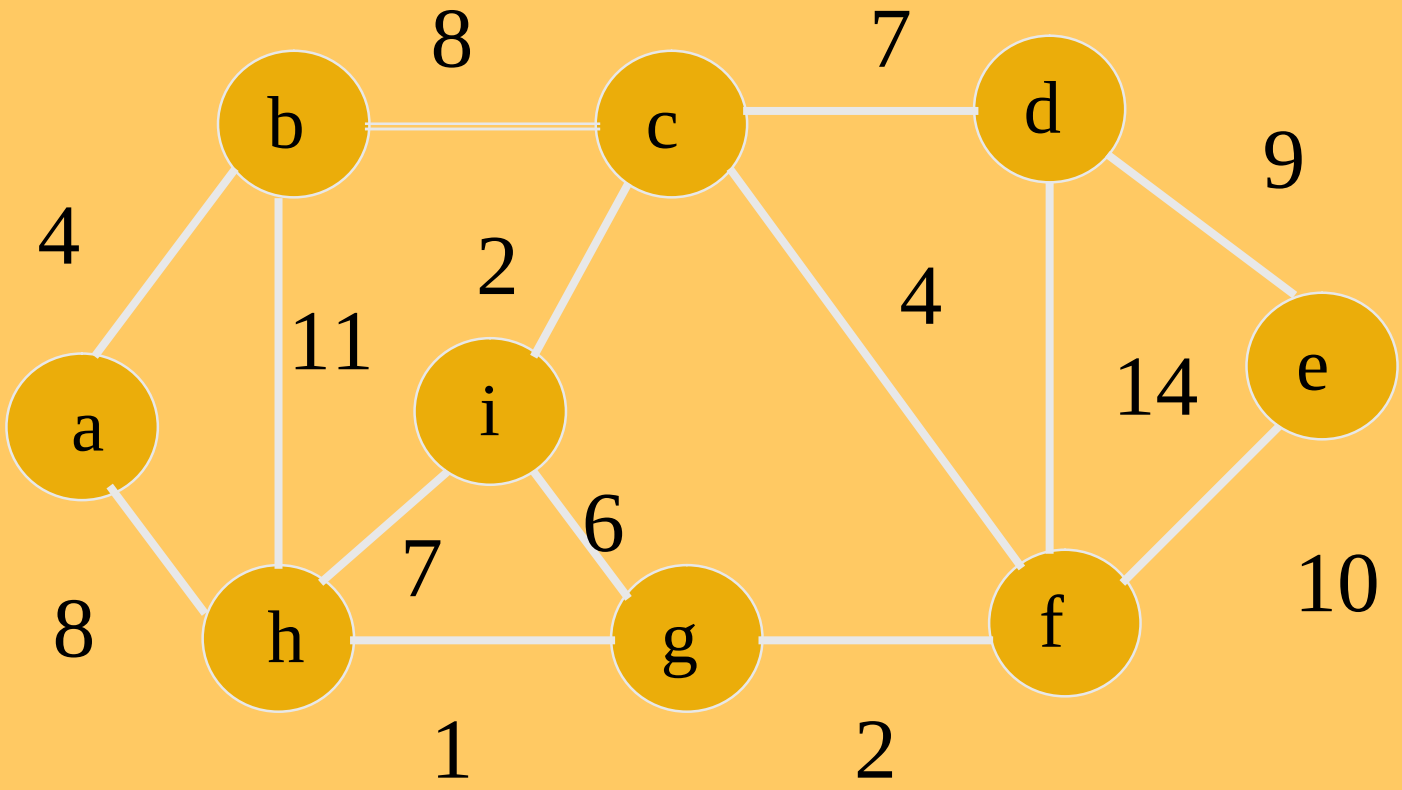
Implementation

- How to determine whether an edge connects a tree node and a non-tree node?
  - We must remember the edges on the boundary, and put them in a heap so that we do the following FAST.
    - Insert new edges
    - Delete the minimum

Kruskal's Algorithm

- Kruskal-MST(*G*)
  - Put the edges in a priority queue ordered by weight.
  - *count*=0
  - while (*count* < *n*-1) do
    - get next edge (*v*,*w*)
    - if (*v* and *w* are not in the same component)
      - Add (*v*,*w*) to the spanning tree.
      - merge component(*v*) and component(*w*)

Kruskal's Algorithm



Property

- We can always find a MSP that contains the edge of the minimum cost (if it is unique).
  - A simple greedy method.
- In addition, we can safely add the minimum cost edges as long as it does not introduce cycle.

Implementation

- How to determine if v and w are in the same component.
- How do we merge two components?

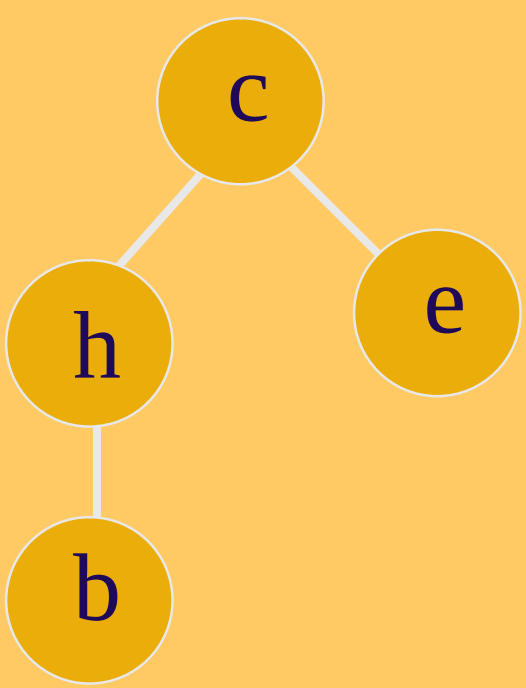
Union and Find

- Answer the two previous problems.
- A powerful data structure to store the “group” information in a tree.
- Each group is a tree. Two nodes are in the same group if they trace to the same root.

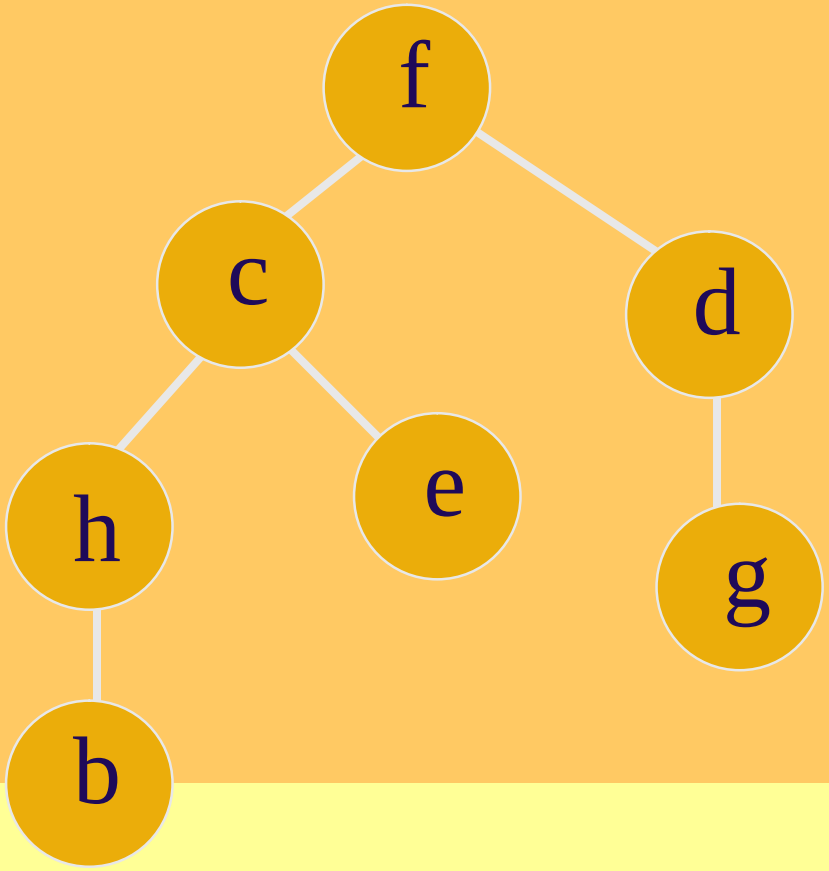
The Union and Find Algorithm

- Find(v)
  - Trace to the root, and return the root.
  - “Flatting” the tree along the way.
- Union(v, w)
  - v, and w do a find respectively.
  - Make the root of v a child of w.

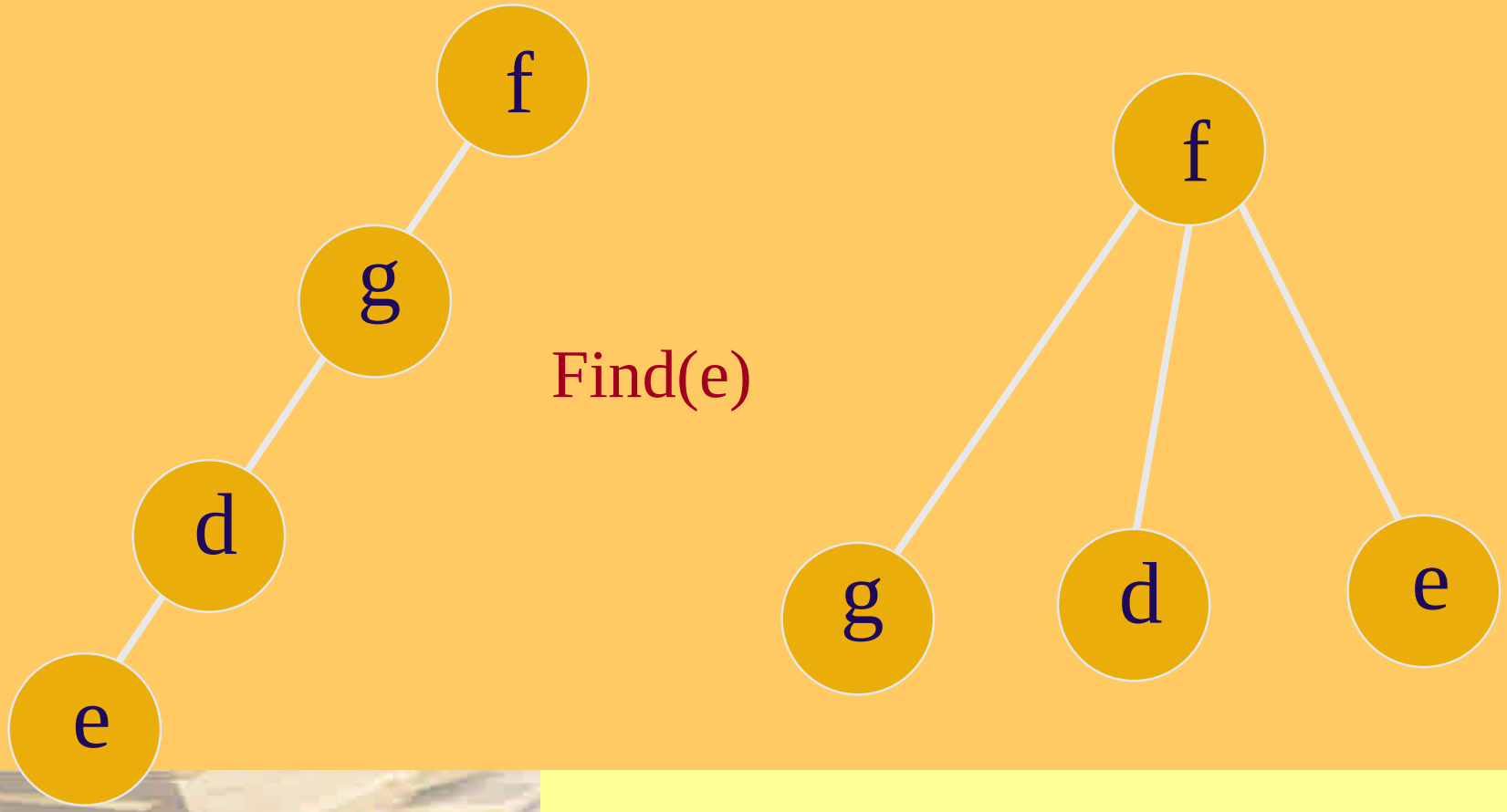
Union(e, g)



Union(e, g)



Find(e)



Kruskal's Algorithm

- How to determine if v and w are in the same component?
  - To detect cycle
  - If v and w are in the same group, the new edge for a cycle.
- How do we merge two components?
  - Union them.

Comparison

- Prim’s algorithm
  - $O(n^2)$
  - Better when the graph is dense.
- Kruskal's Algorithm
  - O(m log m) time if union and find is used.
  - Better when the graph is sparse.

Shortest Path

- Find a minimum path from the given source to all the other nodes.
- Un-weighted
  - All the edges is of unit cost.
  - Can be solved by a BFS.
- Weighted
  - Every edge has its own weight.

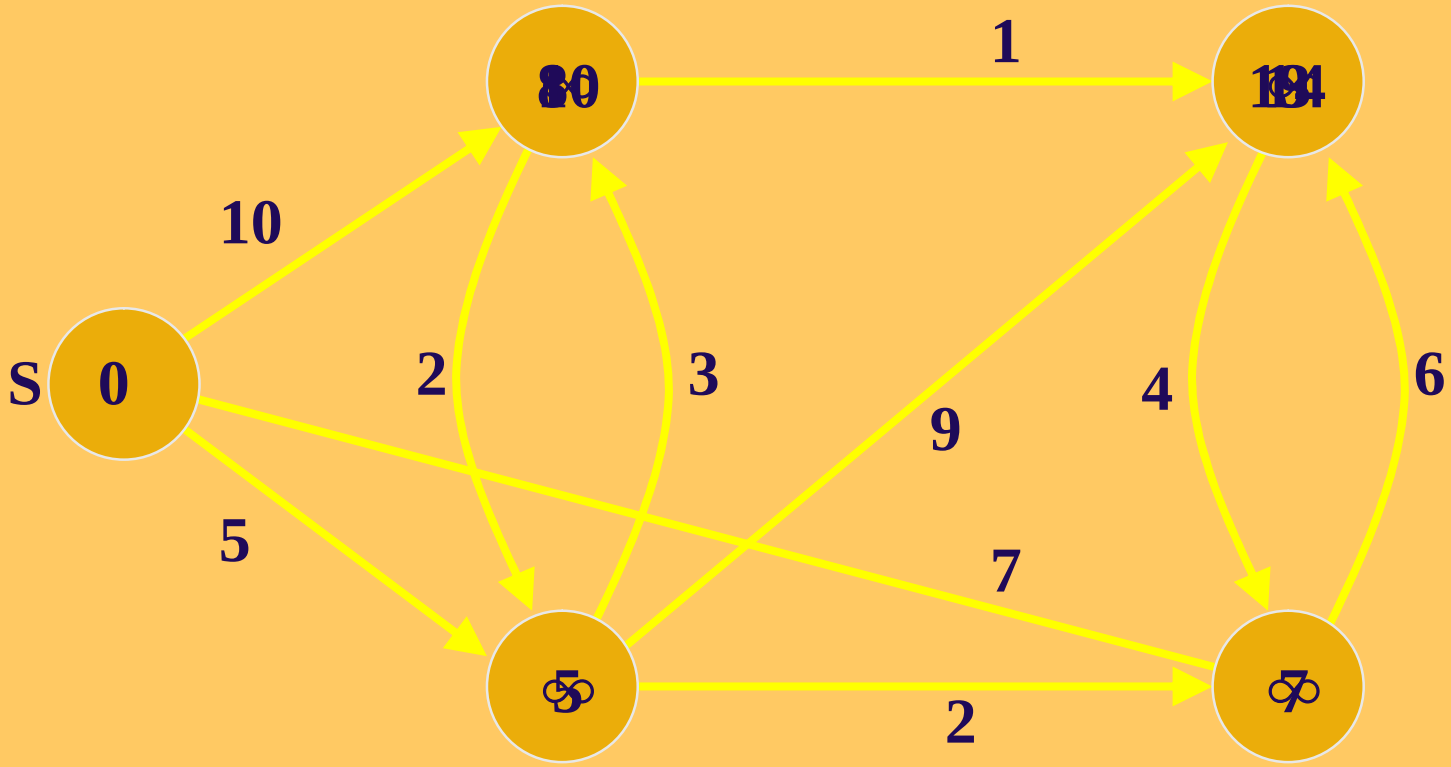
Dijkstra's Algorithm

- No negative edges allowed.
- A dynamic programming approach.
- Maintain a set S of nodes whose shortest path from s have been determined.
- Find a node not in S, and update the distance estimate to all the others.

Dijkstra's Algorithm

- Put s into S
- While S is not V
  - Find the minimum d(v) for node v in V-S
  - Add v into S
  - For every node u adjacent to v
    - d(u) = min(d(u), d(v) + dist(v, u))

Example



Implementation

- Need to maintain a priority of nodes in V – S, with distance estimate as the key.

All Source Shortest Path

- Find the shortest from all the nodes.
- Run Dijkstra's algorithm for every node?
  - Not efficient
- Use a dynamic programming approach.

Floyd's Algorithm

- Let D[i,j,k] be the shortest distance from i to j using the first k nodes as intermediate nodes.
- D[i,j,k] =
  - min (D[i,j,k-1], D[i,k,k-1]+D[k,j,k-1])

The Code

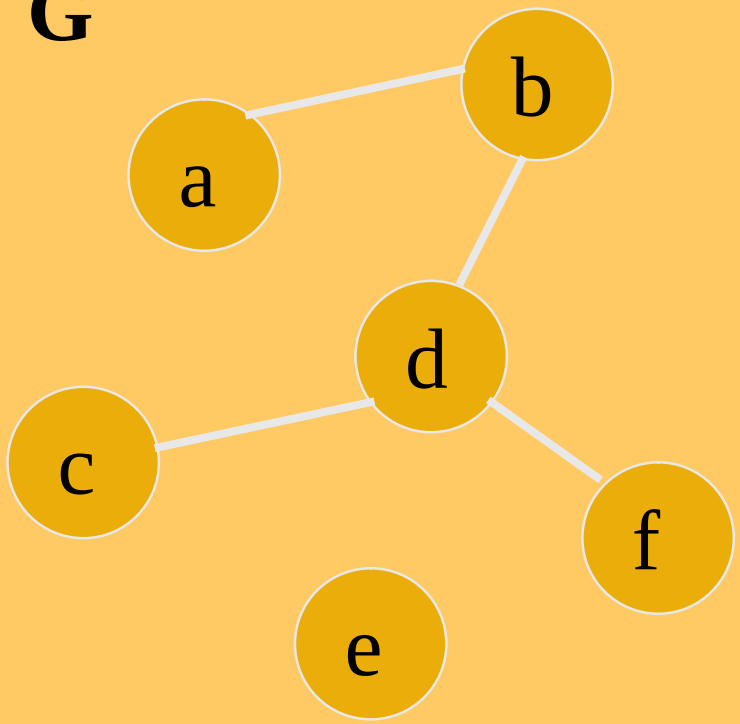
- for k=1 to n
  - for i=1 to n
    - for j=1 to n
      - min (D[i,j,k-1], D[i,k,k-1]+D[k,j,k-1])

Transitive Closure

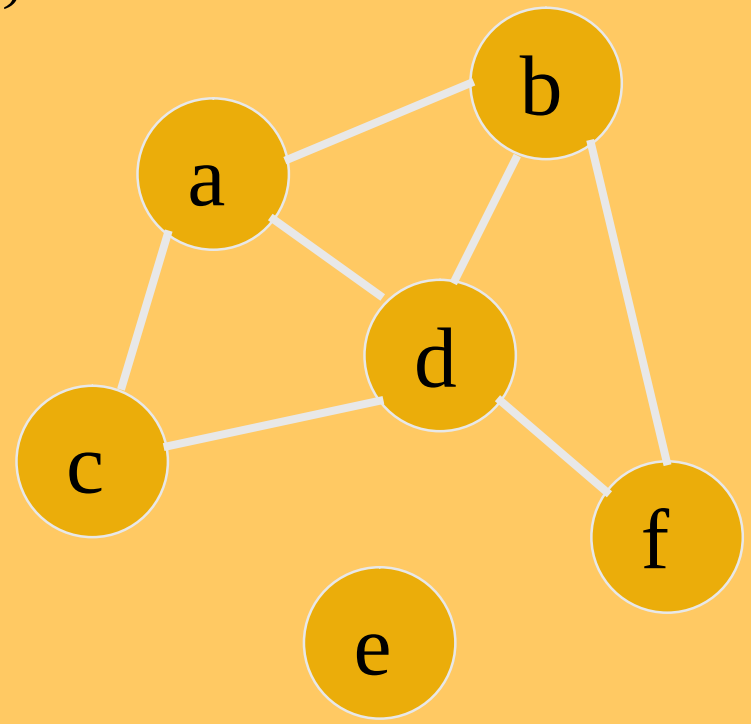
- The transitive closure of a graph G =(V,E) is a graph G’ = (V’, E’), such that
  - V’ = V
  - There is an edge (u, v) in E’ if and only if there is a path from u to v in G.

Transitive Closure

G



G’



Approaches

- BFS or DFS from all sources.
- Strongly connected component.
  - Two nodes are in the same strongly connected components if they reach one another.
  - Reduce the graph in to strongly connected components.

Warshall's Algorithm

- A dynamic programming approach similar to Floyd's all pair shortest path.
- Multiply the adjacent matrix by itself.
  - The inner product is done in Boolean algebra.
    - Only has 1 or 0.
  - $a(i,j) = or_k(a(i,k) \text{ and } (k,j))$

The Code

- for c=1 to log n
  - for *i*=1 to *n*
    - for *j*=1 to *n*
      - for *k*=1 to *n*
        - a(i,j) |= (a(i,k) and (k,j))

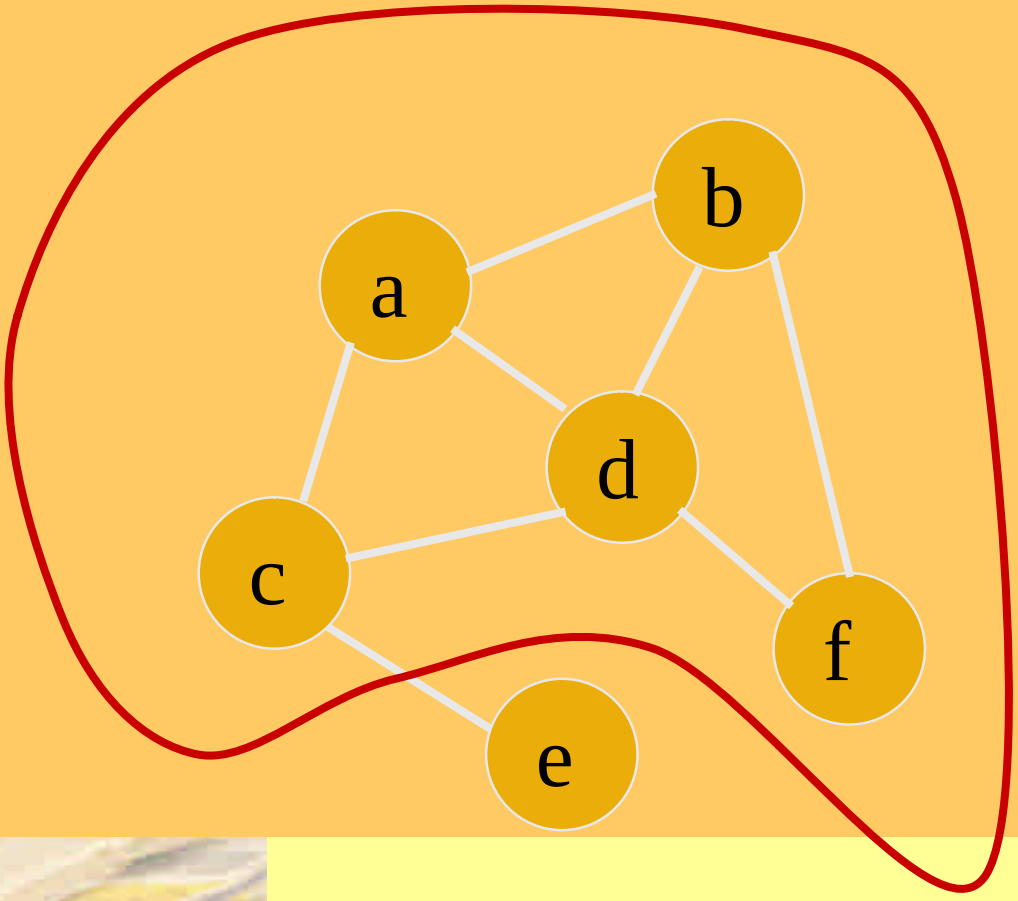
Transitive Reduction

- Given the graph G', find a graph G that doing transitive closure on G yields G'.
- It is like eliminating unnecessary edges from G.

A Quick and Dirty Solution

- Find all strongly connected components and replace them with loops.
- Add those edges that go from one component to another.
- Might add edges that were not in the graph.

An Example



Difficulty

- If we are not allowed to add new edges, it becomes a very difficult problem.
- If there is only one strongly connected component, we must find a Hamiltonian cycle.

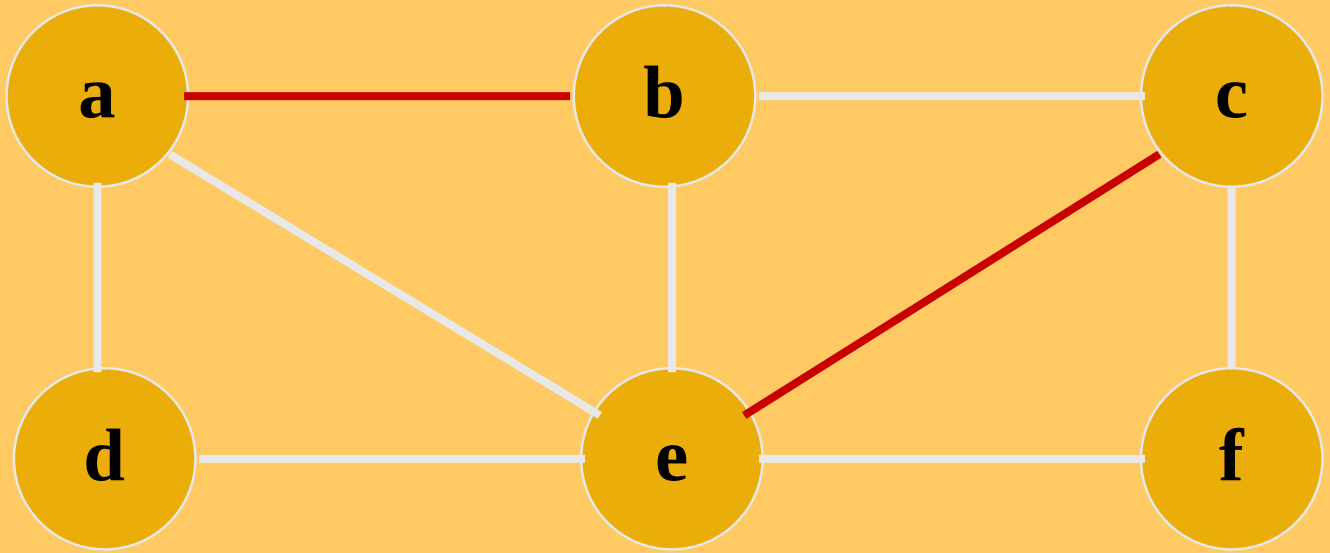
Matching

- A subset of edges that do not share nodes.
- A maximum matching is the match of maximum cardinality.
- A maximal matching is a matching that you cannot add another edge without sharing nodes.

Berge's Theorem

- An augmenting path *P* is a path of edges where every odd-numbered edge (including the first and last edge) is not in *M*, while every even-numbered edge is.
- A matching is maximum if and only if it does not contain any augmenting path.

Matching



Matching M ={(a, b) ,(c, e)} of size 2

path Augmenting P = {(d, a), (a, b), (b, c), (c, e), (e, f)}

Weighted Matching

- Every edge has a weight, and you need to maximize the total weight.
- The famous but mysterious $O(n^3)$ Hungarian method.

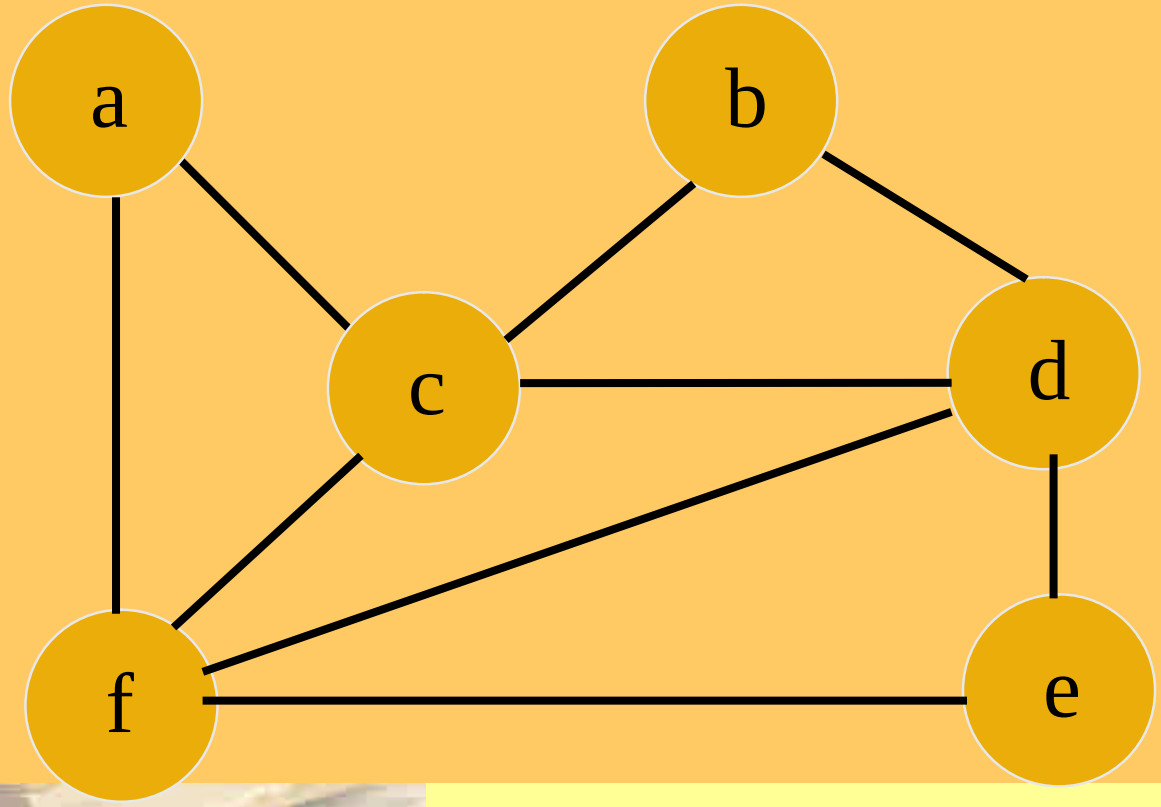
Eulerian Cycle

- Go through every and all edges of a graph exactly once.
- Different from Hamiltonian cycle, which goes through every and all nodes exactly once.

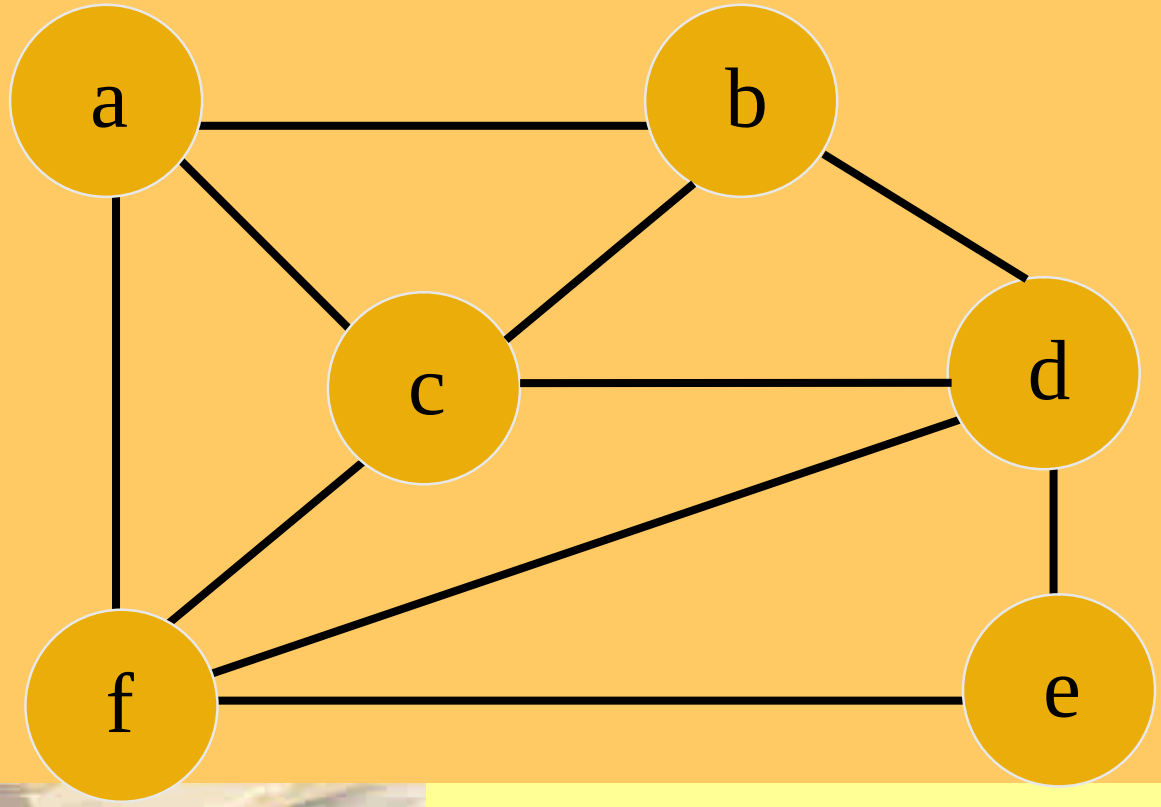
Necessary and Sufficient Cond.

- Eulerian cycle
  - Every node has even degree.
    - If directed, in-degree must be equal to out-degree.
- Eulerian path
  - Every node (except the source and destination) has even degree.
    - If directed, the source's out-degree is equal to in-degree plus 1, the destination's out-degree is equal to in-degree minus 1, and other nodes have the same out-degree and in-degree.

Eulerian cycle



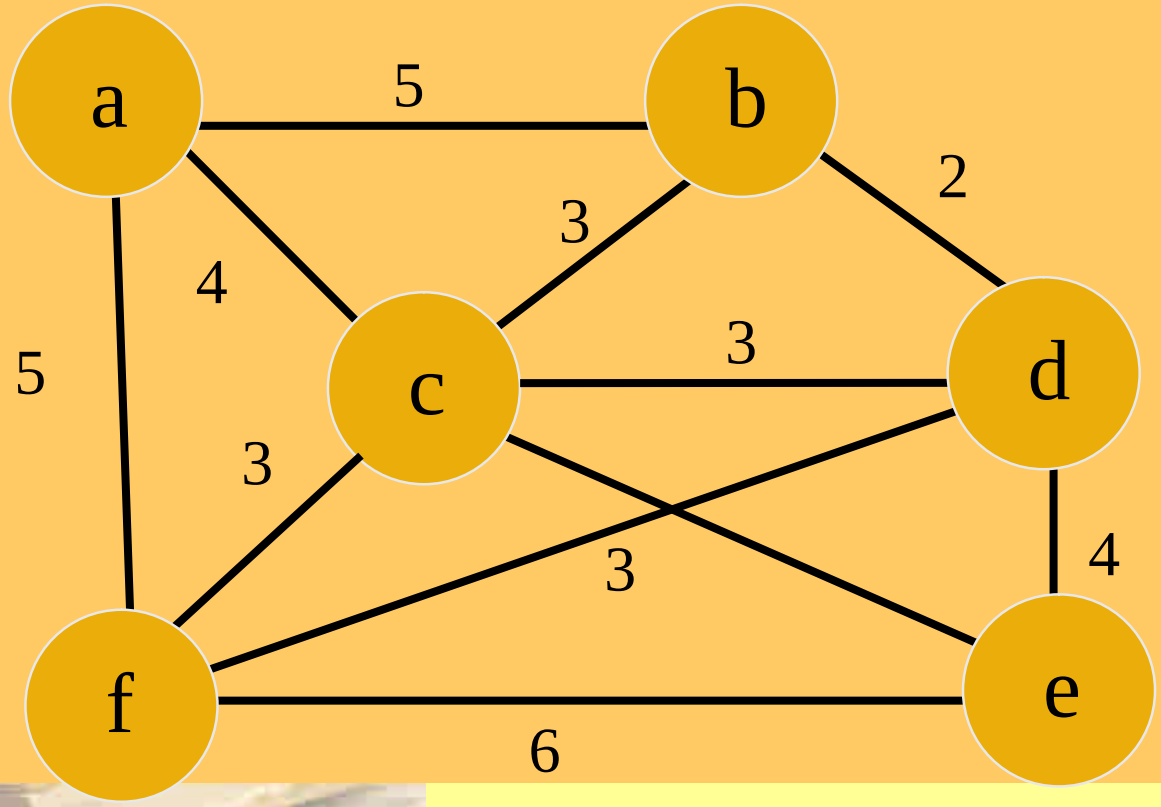
Eulerian path



Postman Tour

- Go through every and all edges of a graph *at least* once.
- It can be shown that no edge will be traversed more than twice.

Postman Tour



The Algorithm

- Find the shortest path between each pair of odd-degree nodes in *G*.
  - This is equivalent to finding a minimum matching among the odd-degree nodes.
- The rest can be done in an Eulerian cycle.

Maximum Flow

- Given a directed graph G= (V,E), and a capacity on every edge. Find the maximum amount of flow that can go from source (nodes without incoming edges) to sinks (nodes without outgoing edges), while respecting the flow capacity constraints.

Maximum Flow