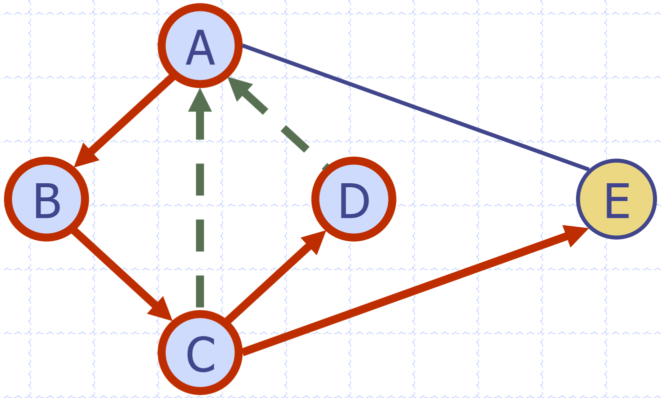
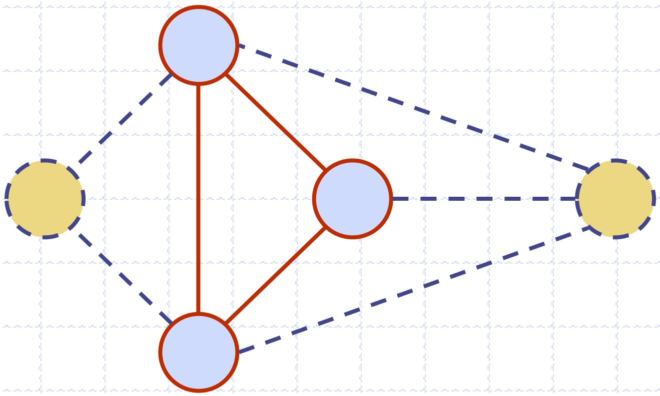
Presentation for use with the textbook Data Structures and Algorithms in Java, 6th edition, by M. T. Goodrich, R. Tamassia, and M. H. Goldwasser, Wiley, 2014

Depth-First Search

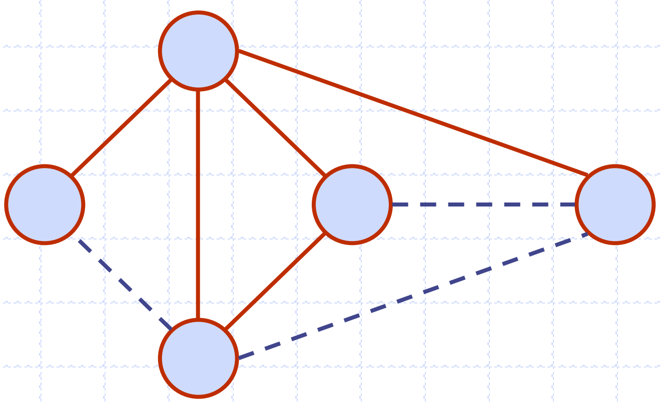


Subgraphs

- A subgraph S of a graph G is a graph such that
  - The vertices of S are a subset of the vertices of G
  - The edges of S are a subset of the edges of G
- A spanning subgraph of G is a subgraph that contains all the vertices of G



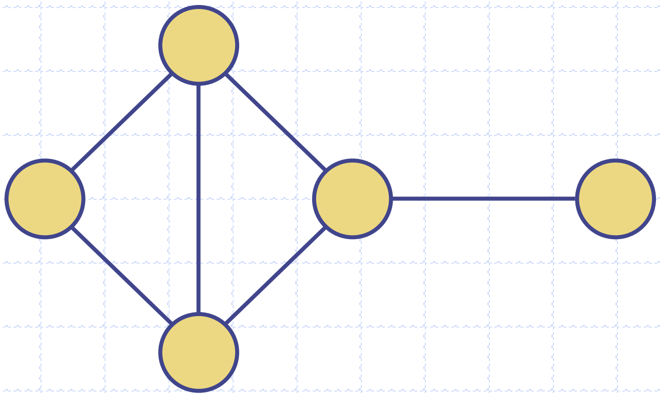
Subgraph



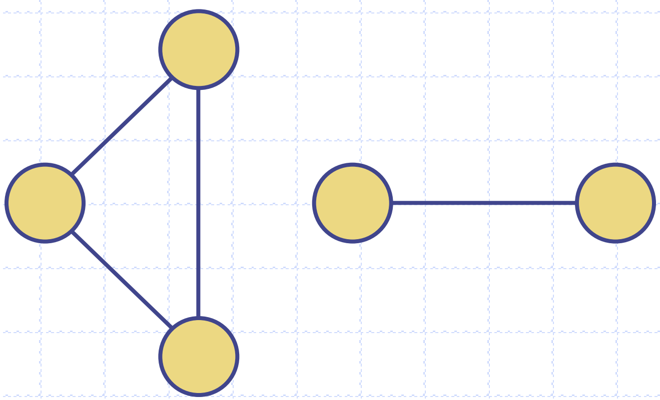
Spanning subgraph

Connectivity

- A graph is connected if there is a path between every pair of vertices
- A connected component of a graph G is a maximal connected subgraph of G



Connected graph

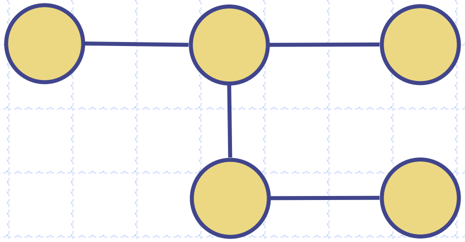


Non connected graph with two connected components

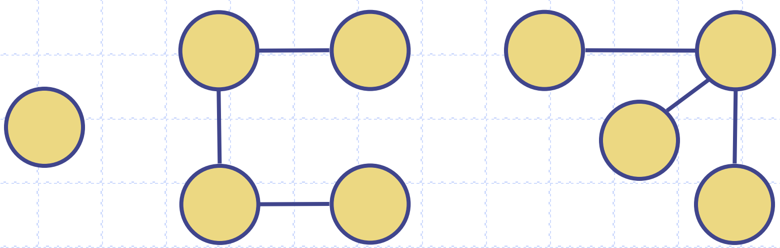
Trees and Forests

- A (free) tree is an undirected graph T such that
  - T is connected
  - T has no cycles

  This definition of tree is different from the one of a rooted tree
- A forest is an undirected graph without cycles
- The connected components of a forest are trees



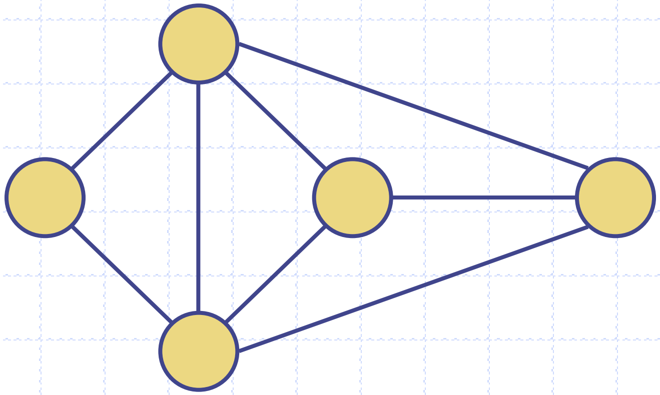
Tree



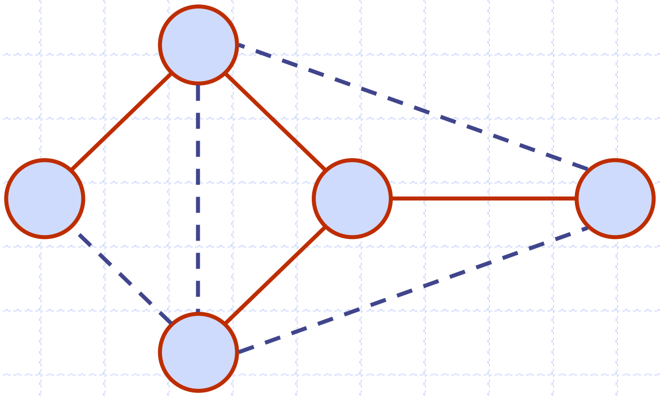
Forest

Spanning Trees and Forests

- A spanning tree of a connected graph is a spanning subgraph that is a tree
- A spanning tree is not unique unless the graph is a tree
- Spanning trees have applications to the design of communication networks
- A spanning forest of a graph is a spanning subgraph that is a forest



Graph



Spanning tree

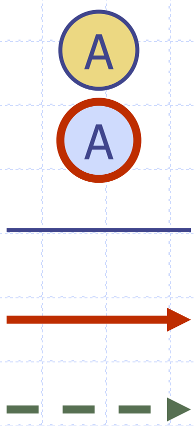
Depth-First Search

- Depth-first search (DFS) is a general technique for traversing a graph
- A DFS traversal of a graph G
  - Visits all the vertices and edges of G
  - Can determine whether G is connected
  - Can compute the connected components of G
  - Can computes a spanning forest of G
- DFS can be further extended to solve other graph problems
  - Find and report a path between two given vertices
  - Find a cycle in the graph

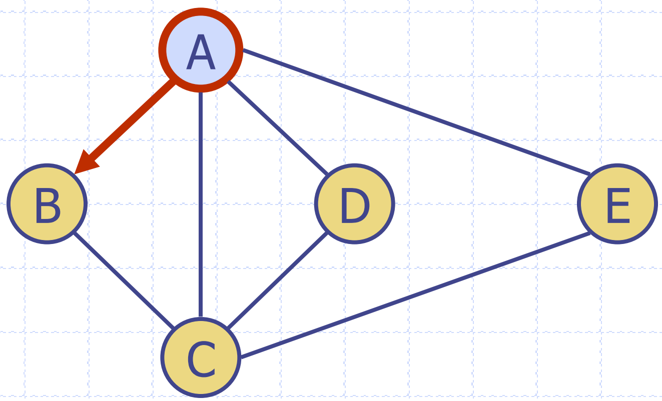
DFS Algorithm from a Vertex

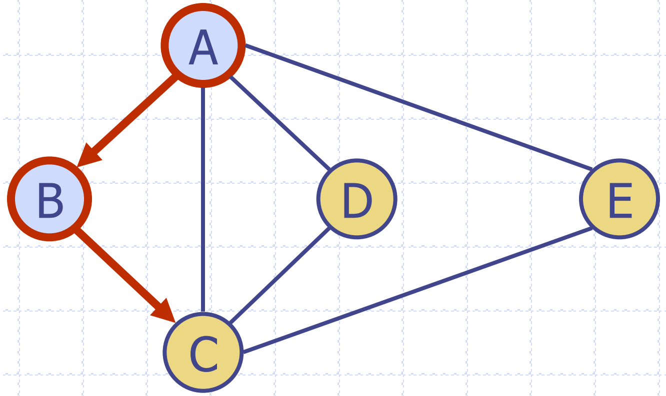
```
Algorithm DFS(u)
In: Vertex u of a graph G
Out: {DFS traversal of G starting at u}

Mark (u)
For each edge (u,v) incident on u do
    if (u,v) is not labelled then
        if v is not marked then {
            Label (u,v) as "discovery edge"
            DFS(v)
        }
        else label (u,v) as "back edge"
```

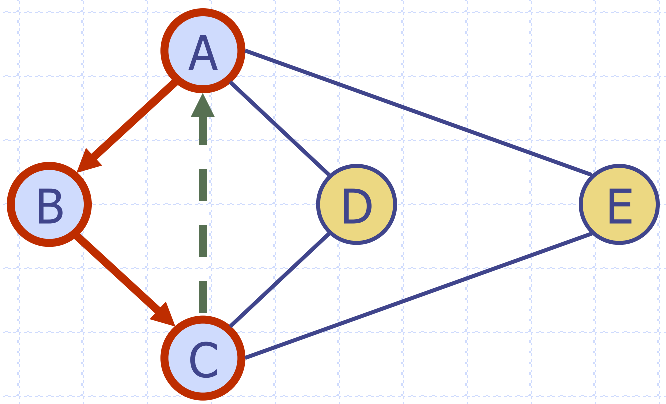
Example



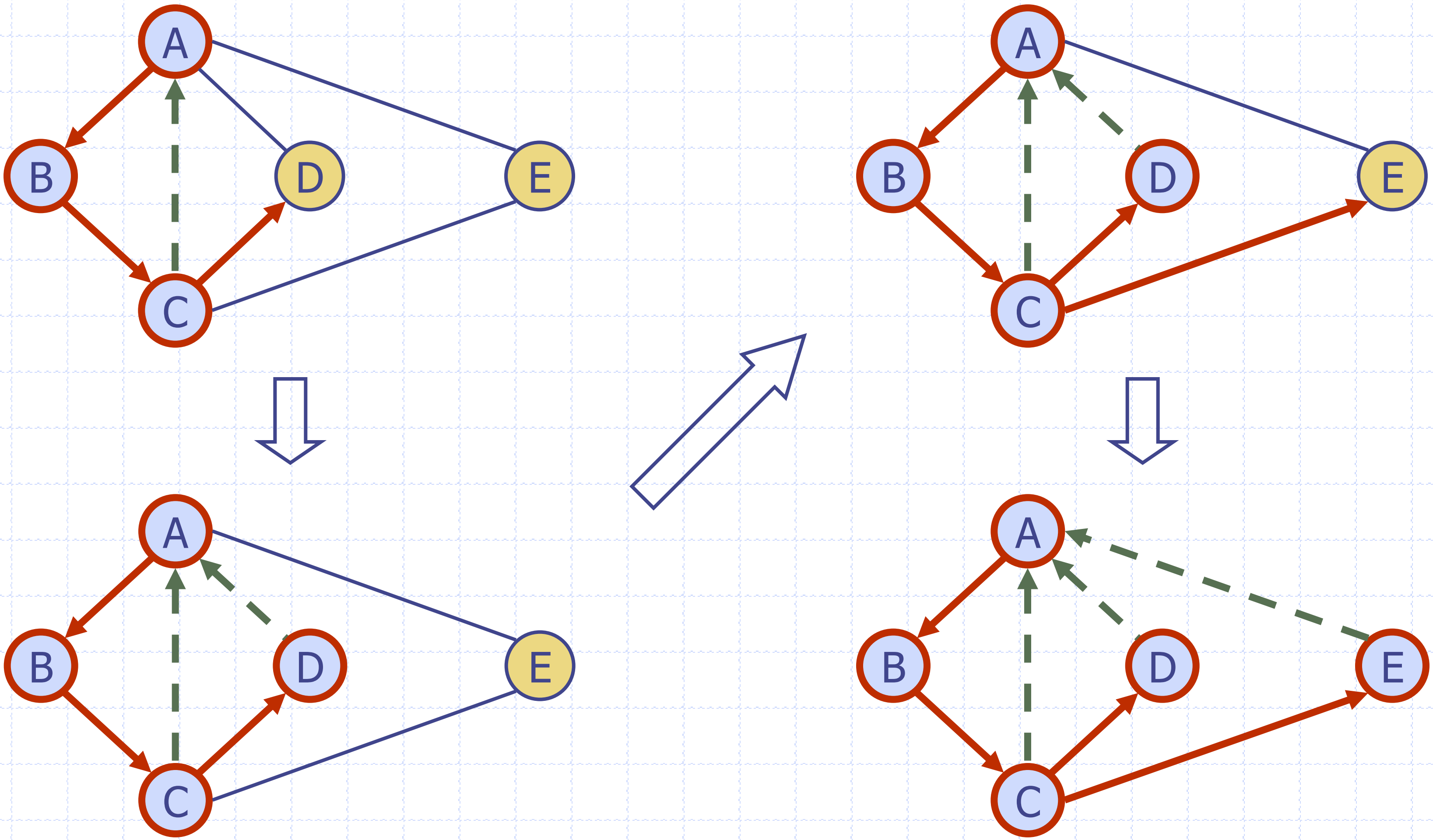
unexplored vertex
visited vertex
unexplored edge
discovery edge
back edge



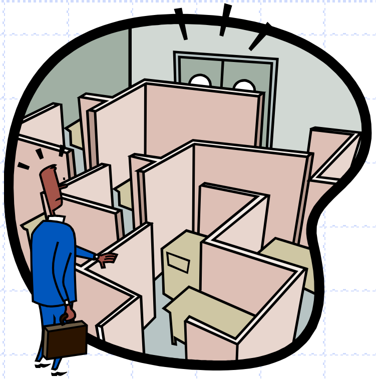




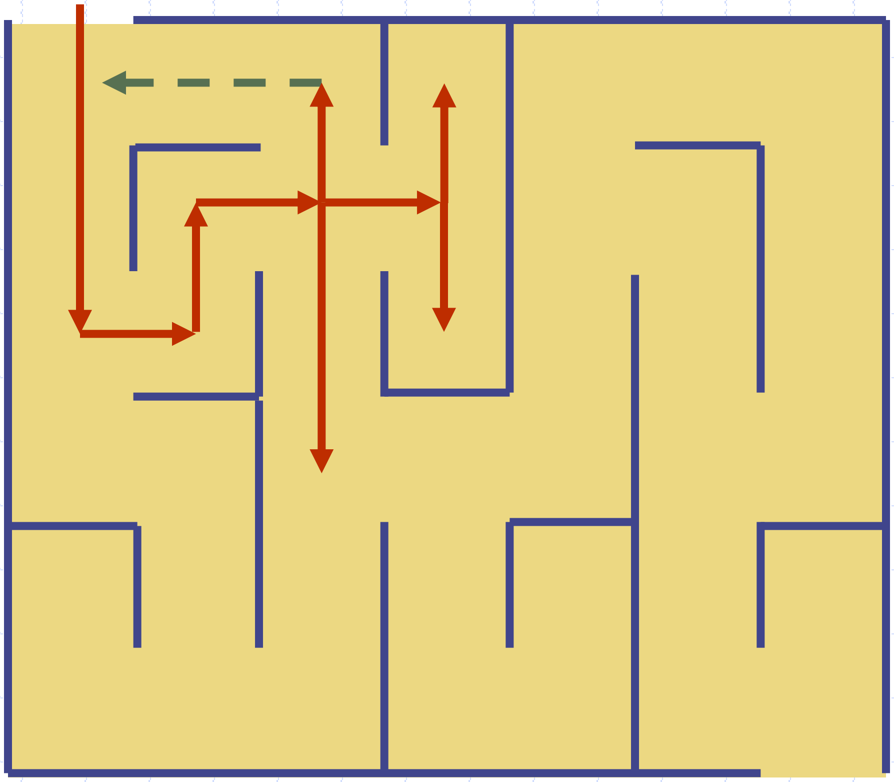
Example (cont.)



DFS and Maze Traversal

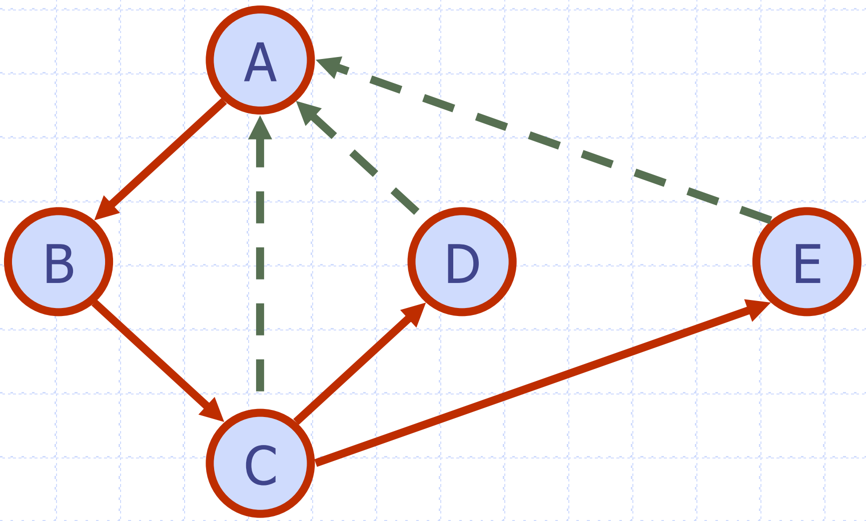


- The DFS algorithm is similar to a classic strategy for exploring a maze
  - We mark each intersection, corner and dead end (vertex) visited
  - We mark each corridor (edge) traversed
  - We keep track of the path back to the entrance (start vertex) by means of a rope (recursion stack)

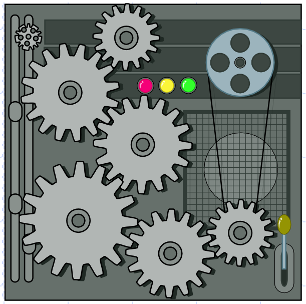


Properties of DFS

- Property 1
  - ***DFS*(*G*, *v*)** visits all the vertices and edges in the connected component of ***v***
- Property 2
  - The discovery edges labeled by ***DFS*(*G*, *v*)** form a spanning tree of the connected component of ***v*** called a ***DFS* tree**



Analysis of DFS



- Setting/getting a vertex/edge label takes $O(1)$ time
- Each vertex is labeled twice
  - once initialized as UNEXPLORED
  - once as VISITED
- Each edge is labeled twice
  - once initialized as UNEXPLORED
  - once as DISCOVERY or BACK
- Method incidentEdges is called once for each vertex
- DFS runs in $O(n + m)$ time provided the graph is represented by the adjacency list structure and it runs in $O(n^2)$ time if the graph is stored in an adjacency matrix.

Path Finding

- We can specialize the DFS algorithm to find a path between two given vertices u and z
- We call $DFS(u)$ with u as the start vertex
- We use a stack S to keep track of the path between the start vertex and the current vertex
- As soon as destination vertex z is encountered, we return the path as the contents of the stack

```
Algorithm pathDFS(G, v, z)
    Mark(v)
    S.push(v)
    if v = z
        return true
    for all edges (v, w) incident on v do
        if w is not marked then
            if pathDFS(G, w, z) then
                return true
    S.pop(v)
    return false
```