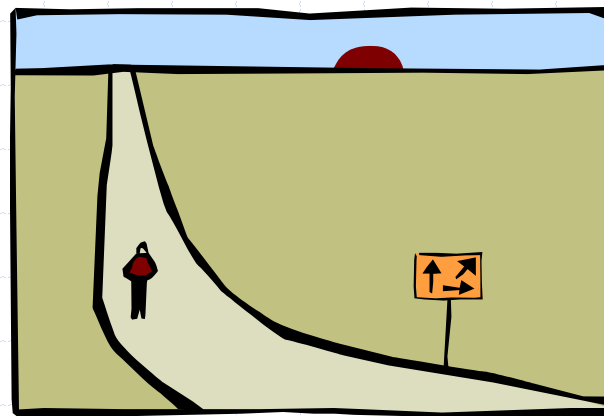


# 그래프 순회



# Outline

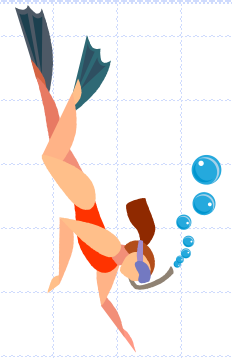
- ◆ 14.1 그래프 순회
- ◆ 14.2 깊이우선탐색
- ◆ 14.3 너비우선탐색
- ◆ 14.4 응용문제

# 그래프 순회

- ◆ 순회(traversal): 모든 정점과 간선을 검사함으로써 그래프를 탐험하는 체계적인 절차
- ◆ 순회 예
  - 수도권 전철망의 모든 역(정점)의 위치를 출력
  - 항공사의 모든 항공편(간선)에 대한 노선 정보를 수집
  - 웹 검색엔진의 데이터 수집 부문은 웹의 하이퍼텍스트 문서(정점)와 문서 내 링크(간선)를 검사함으로써 서핑
- ◆ 주요 전략
  - 깊이우선탐색
  - 너비우선탐색



# 깊이우선탐색



◆ **깊이우선탐색**(depth-first search, **DFS**):  
그래프를 순회하기  
위한 일반적 기법

◆ 그래프  $G$ 에 대한 **DFS**  
순회로 가능한 것들

- $G$ 의 모든 정점과 간선을 방문
- $G$ 가 연결그래프인지 결정
- $G$ 의 연결요소들을 계산
- $G$ 의 신장숲을 계산

◆  $n$ 개의 정점과  $m$ 개의 간선을 가진 그래프에 대한 **DFS**는  $O(n + m)$  시간 소요

◆ **DFS**를 확장하면 해결 가능한 그래프 문제들

- 두 개의 주어진 정점 사이의 경로를 찾아 보고하기
- 그래프 내 싸이클 찾기

◆ 그래프에 대한 **깊이우선탐색**은 이진트리에 대한 **선위순회**와 유사

# DFS

## Alg *DFS*(*G*)

**input** graph *G*

**output** labeling of the edges of *G* as  
tree edges and back edges

{The algorithm uses a mechanism for  
setting and getting “labels” of vertices  
and edges}

1. **for each** *u*  $\in$  *G.vertices*()  
    *l(u)*  $\leftarrow$  *Fresh*
2. **for each** *e*  $\in$  *G.edges*()  
    *l(e)*  $\leftarrow$  *Fresh*
3. **for each** *v*  $\in$  *G.vertices*()  
    **if** (*l(v)* = *Fresh*)  
        *rDFS*(*G*, *v*)

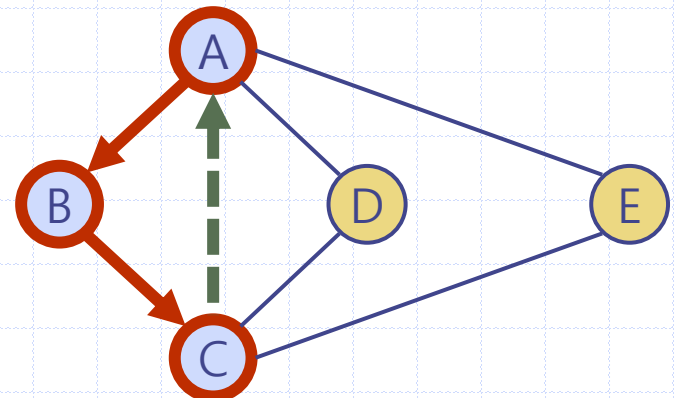
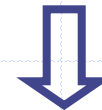
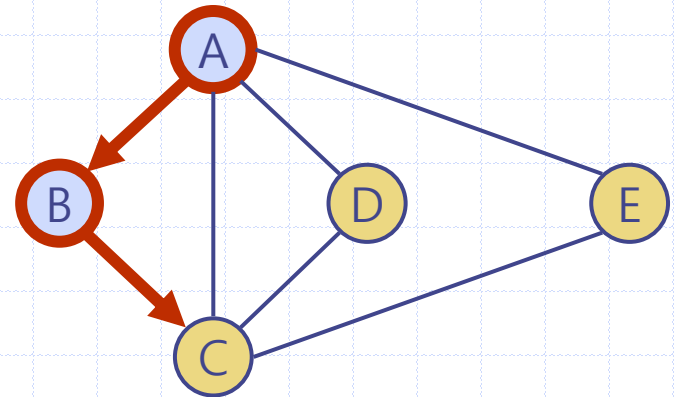
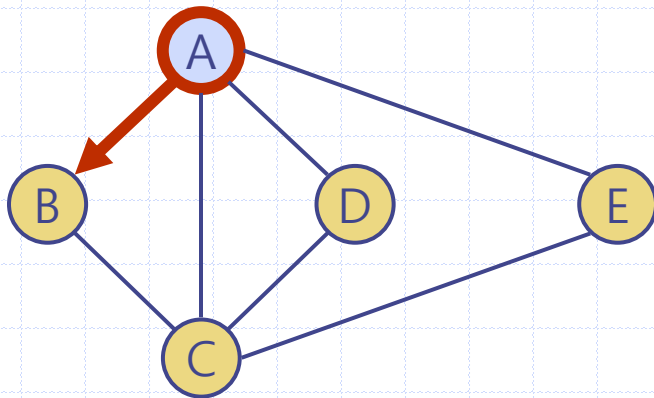
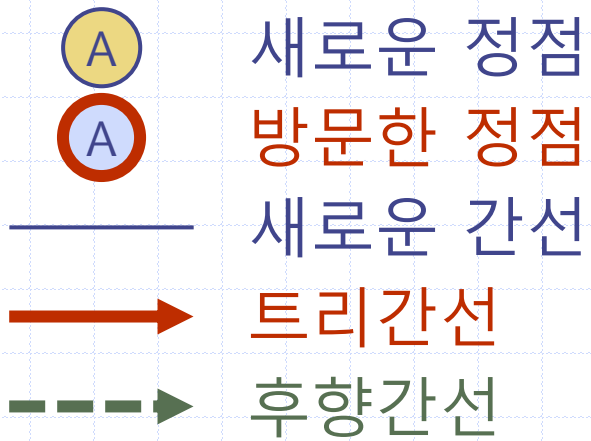
## Alg *rDFS*(*G*, *v*)

**input** graph *G* and a start vertex *v* of  
*G*

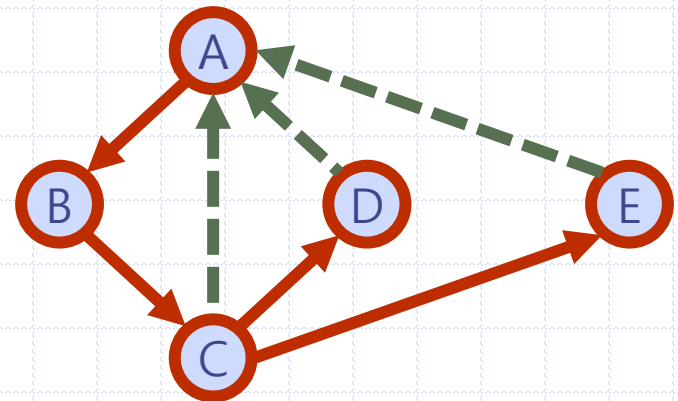
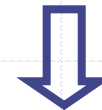
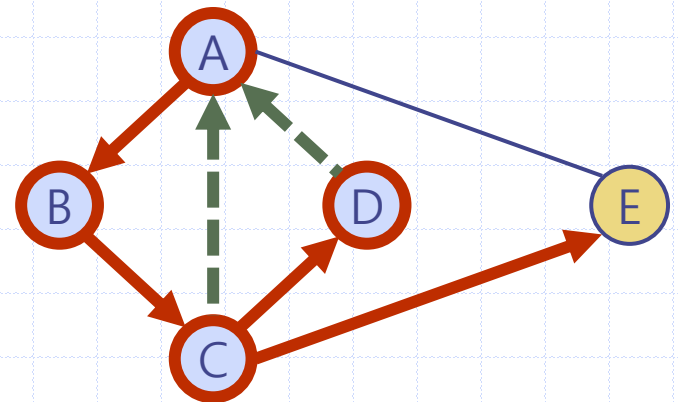
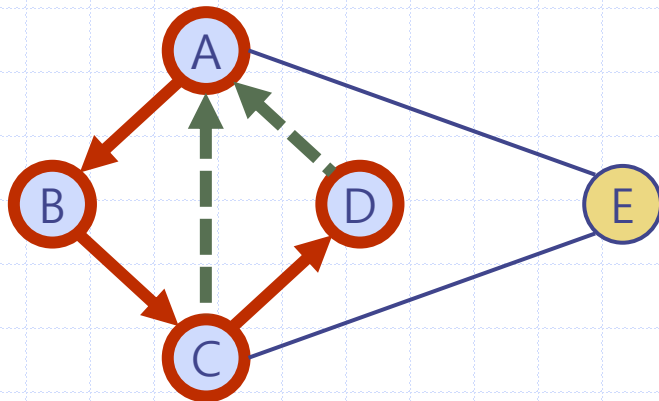
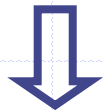
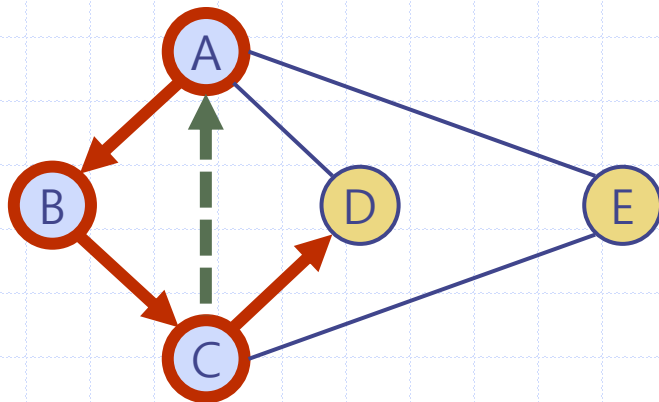
**output** labeling of the edges of *G* in  
the connected component of *v* as  
tree edges and back edges

1. *l(v)*  $\leftarrow$  *Visited*
2. **for each** *e*  $\in$  *G.incidentEdges*(*v*)  
    **if** (*l(e)* = *Fresh*)  
        *w*  $\leftarrow$  *G.opposite*(*v*, *e*)  
        **if** (*l(w)* = *Fresh*)  
            *l(e)*  $\leftarrow$  *Tree*  
            *rDFS*(*G*, *w*)  
        **else**  
            *l(e)*  $\leftarrow$  *Back*

# DFS 수행 예

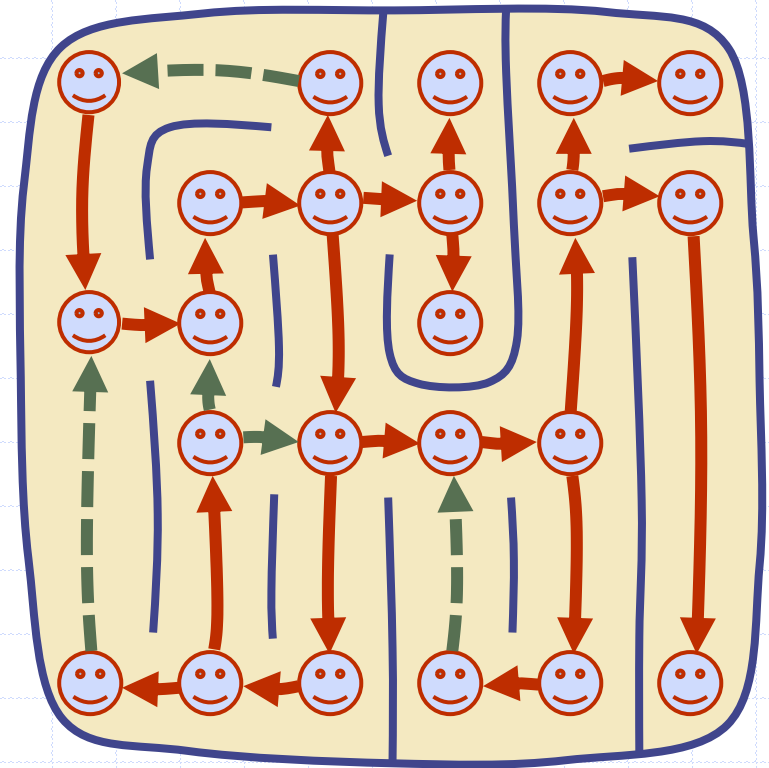


# DFS 수행 예 (conti.)



# DFS와 미로 순회

- ◆ DFS 알고리즘은 미로를 탐험하는데 있어서의 고전적이고 모험적인 전략과 유사
  - 방문한 교차로, 모퉁이, 막힌 복도(모두 정점)를 표시
  - 순회한 복도(모두 간선)를 표시
  - 입구(출발정점)로 되돌아가는 경로를 끈(재귀 스택)을 사용하여 추적





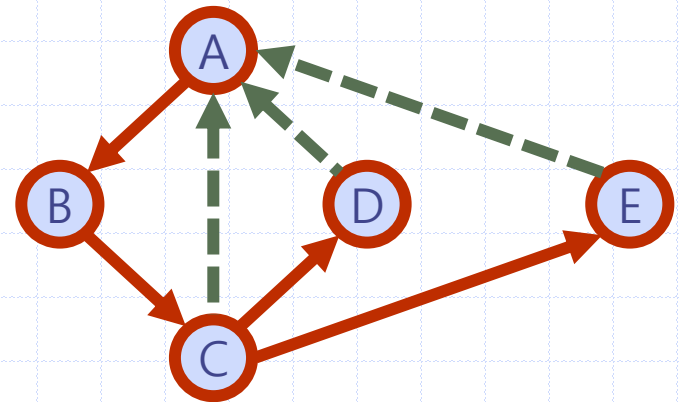
# DFS 속성

## 속성 1

$\text{rDFS}(G, v)$ 는  $v$ 의  
연결요소내의 모든 정점과  
간선을 방문

## 속성 2

$\text{rDFS}(G, v)$ 에 의해 라벨된  
트리 간선들은  $v$ 의  
연결요소의 **신장트리(DFS  
tree**라 불림)를 형성

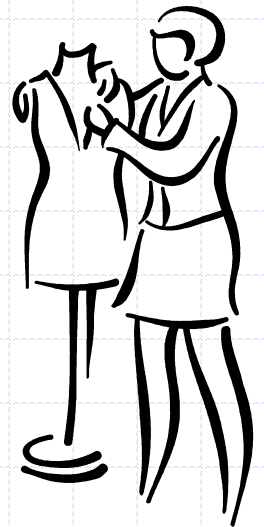


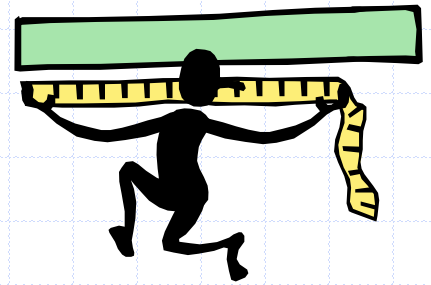
# DFS 분석

- ◆ 정점과 간선의 **라벨**을 쓰고 읽는데  **$O(1)$**  시간 소요
  - 참고: 정점이나 간선을 구현하는 노드 위치의 기능성에 "*Visited*" 플래그를 포함하도록 확장 가능
- ◆ 각 정점은 두 번 라벨
  - 한 번은 *Fresh*로, 또 한 번은 *Visited*로
- ◆ 각 간선은 두 번 라벨
  - 한 번은 *Fresh*로, 또 한 번은 *Tree* 또는 *Back*으로
- ◆ 메소드 **incidentEdges**는 각 정점에 대해 한 번 호출
- ◆ 그래프가 **인접리스트** 구조로 표현된 경우, **DFS**는  **$O(n + m)$**  시간에 수행
  - 참고:  $\sum_v \text{deg}(v) = 2m$

# DFS 템플릿 활용

- ◆ DFS 순회로 더욱 흥미있는 작업을 수행코자 한다면, DFS 알고리즘을 확장해야 한다
- ◆ 이를 DFS 순회 원형 메소드의 **특화**(specialization)라 부른다
- ◆ DFS를 확장하여 해결할 수 있는 문제의 예
  - 연결성 검사
  - 경로 찾기
  - 사이클 찾기





# 너비우선탐색

◆ 너비우선탐색(breadth-first search, **BFS**):  
그래프를 순회하기  
위한 일반적 기법

◆ 그래프  $G$ 에 대한 **BFS**  
순회로 가능한 것들

- $G$ 의 모든 정점과 간선을 방문
- $G$ 가 연결그래프인지 결정
- $G$ 의 연결요소들을 계산
- $G$ 의 신장숲을 계산

◆  $n$ 개의 정점과  $m$ 개의 간선을 가진 그래프에 대한 **BFS**는  $O(n + m)$  시간 소요

◆ **BFS**를 확장하면 해결 가능한 그래프 문제들

- 두 개의 주어진 정점 사이의 최소 간선을 사용하는 경로를 찾아 보고하기
- 그래프 내 단순싸이클 찾기

◆ 그래프에 대한  
**너비우선탐색**은  
이진트리에 대한  
**레벨순회**와 유사

# BFS

## Alg **BFS**( $G$ )

**input** graph  $G$

**output** labeling of the edges of  $G$   
as tree edges and cross edges

{ The algorithm uses a mechanism  
for setting and getting “labels” of  
vertices and edges }

1. **for each**  $u \in G.vertices()$   
     $l(u) \leftarrow Fresh$
2. **for each**  $e \in G.edges()$   
     $l(e) \leftarrow Fresh$
3. **for each**  $v \in G.vertices()$   
    **if** ( $l(v) = Fresh$ )  
         $BFS1(G, v)$

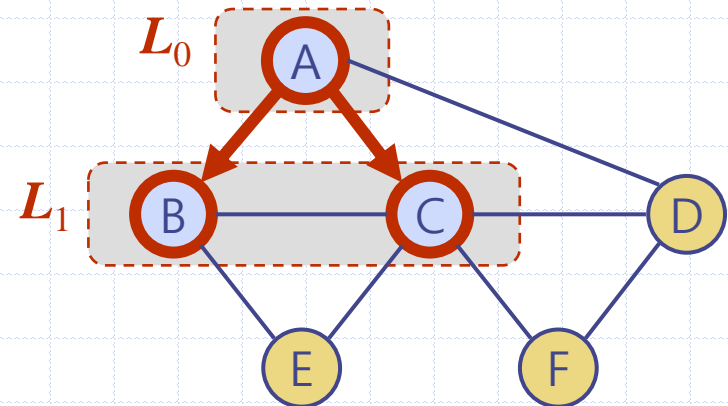
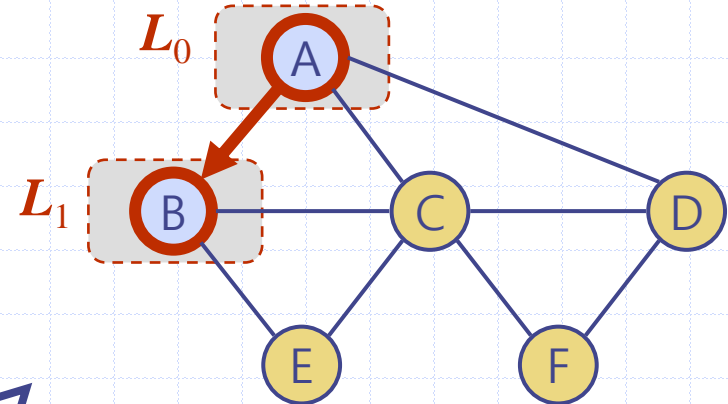
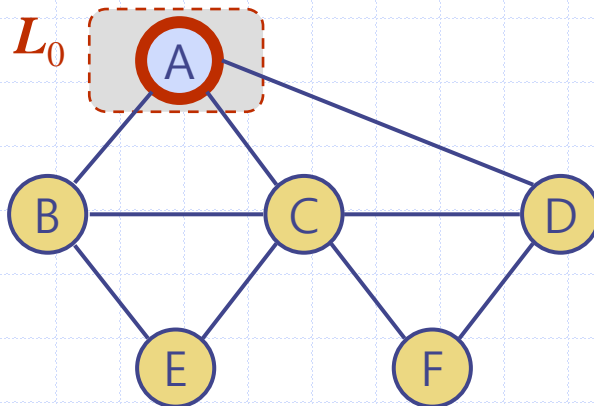
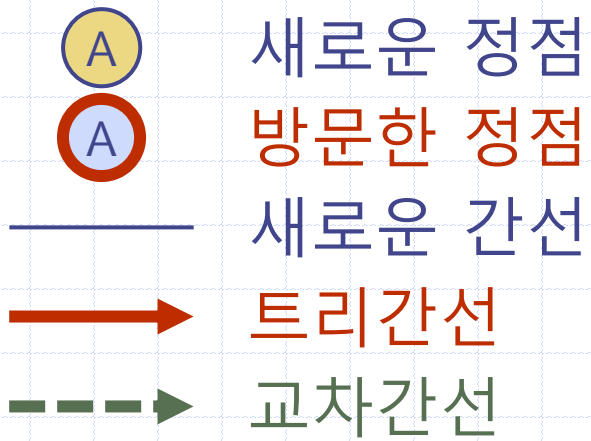
## Alg **BFS1**( $G, v$ )

**input** graph  $G$  and a start vertex  $v$  of  $G$

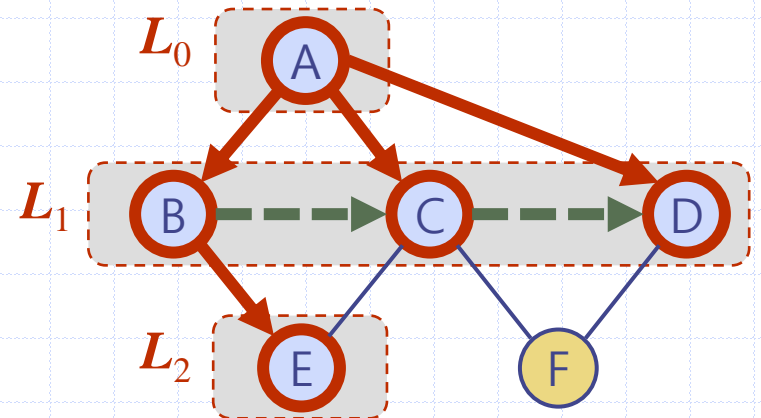
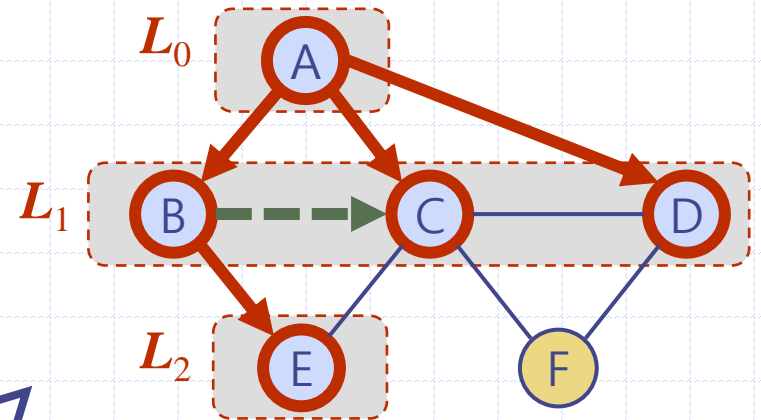
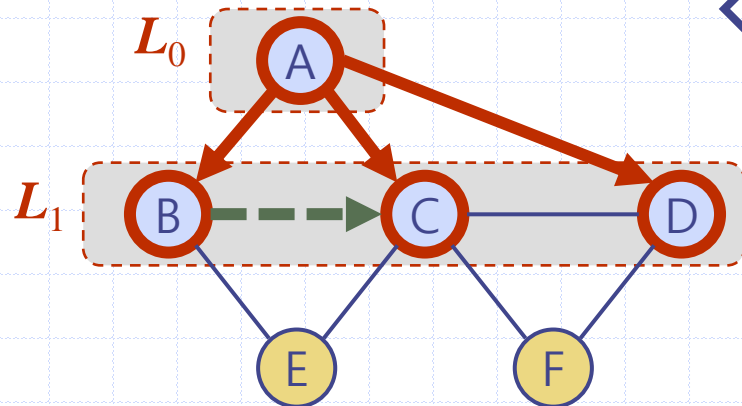
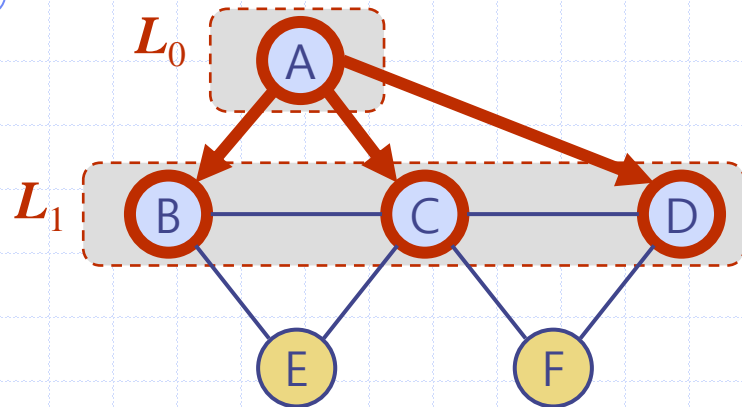
**output** labeling of the edges of  $G$  in the  
connected component of  $v$  as tree edges  
and cross edges

1.  $L_0 \leftarrow empty\ list$       {level container}
2.  $L_0.addLast(v)$
3.  $l(v) \leftarrow Visited$
4.  $i \leftarrow 0$
5. **while** ( $\neg L_i.isEmpty()$ )  
     $L_{i+1} \leftarrow empty\ list$   
    **for each**  $v \in L_i.elements()$   
        **for each**  $e \in G.incidentEdges(v)$   
            **if** ( $l(e) = Fresh$ )  
                 $w \leftarrow G.opposite(v, e)$   
                **if** ( $l(w) = Fresh$ )  
                     $l(e) \leftarrow Tree$   
                     $l(w) \leftarrow Visited$   
                     $L_{i+1}.addLast(w)$   
                **else**  
                     $l(e) \leftarrow Cross$   
     $i \leftarrow i + 1$

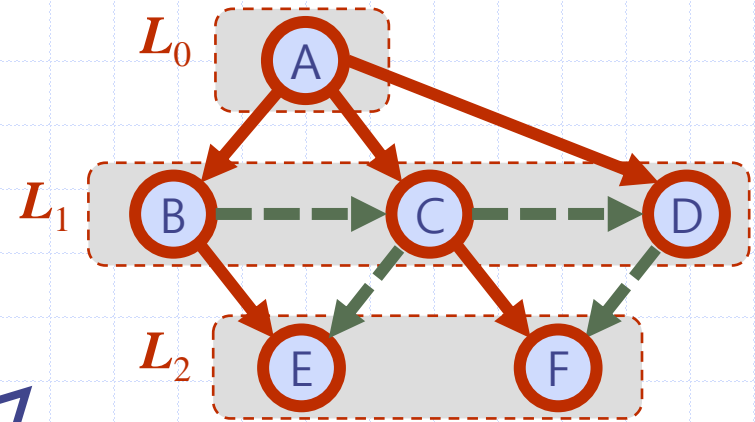
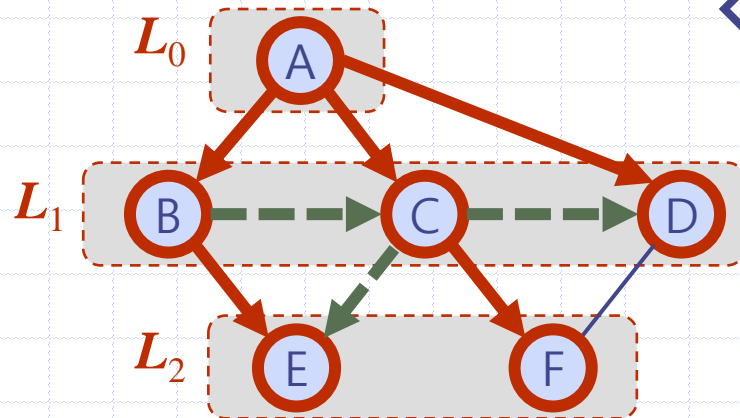
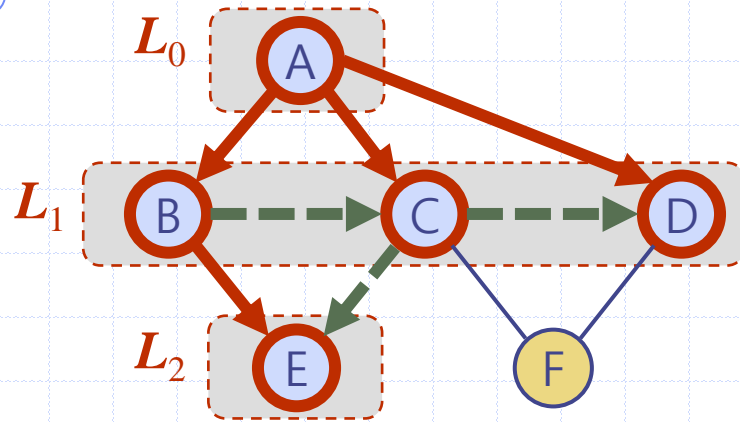
# BFS 수행 예



# BFS 수행 예 (conti.)



# BFS 수행 예 (conti.)

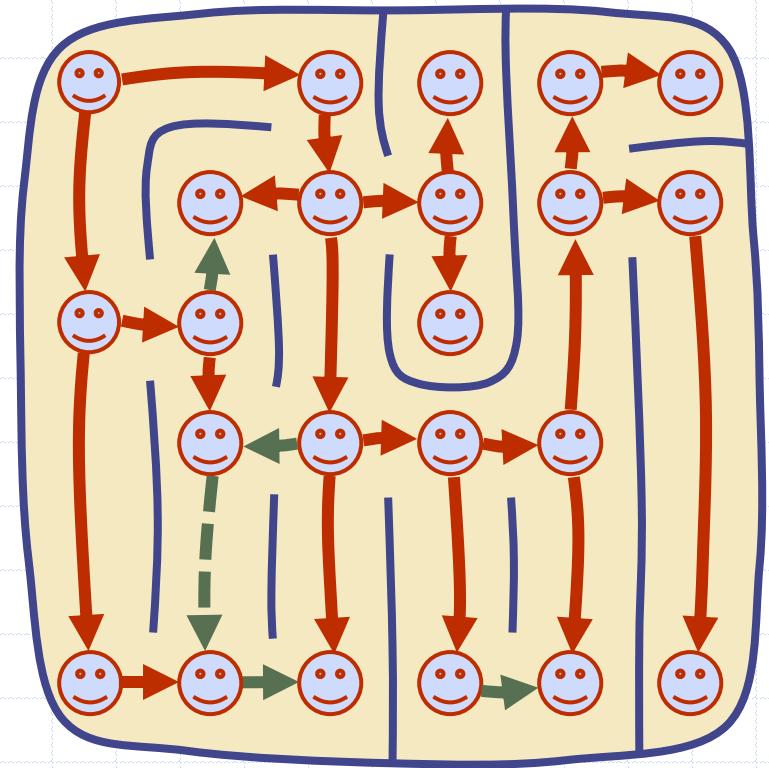




# BFS와 미로 순회

◆ BFS 알고리즘은 미로를 탐험하는데 있어서의 다소 보수적인 전략과 유사

- 방문한 교차로, 모퉁이, 막힌 복도(모두 정점임)를 표시
- 순회한 복도(모두 간선임)를 표시
- 레벨을 하나씩 증가시키면서 진행



# BFS 속성

## 표기

$G_v$ :  $v$ 의 연결요소

## 속성 1

$\text{BFS1}(G, v)$ 는  $G_v$ 의 모든 정점과 간선을 방문

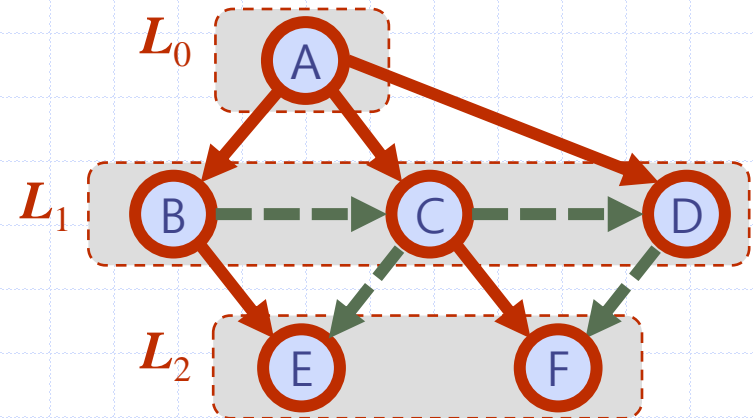
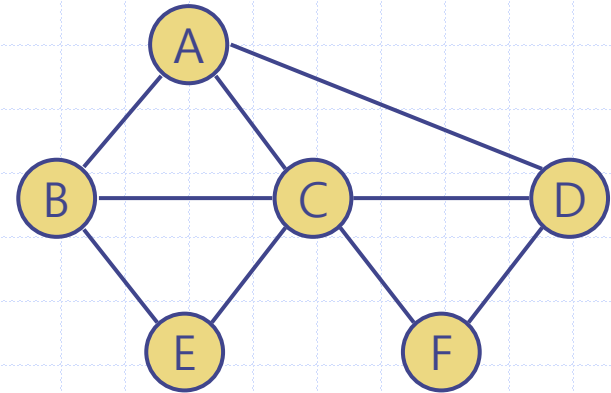
## 속성 2

$\text{BFS1}(G, v)$ 에 의해 라벨된 트리 간선들은  $G_v$ 의 신장트리(BFS tree라 불림)  $T_v$ 를 형성

## 속성 3

$L_i$  내의 각 정점  $w$ 에 대해,

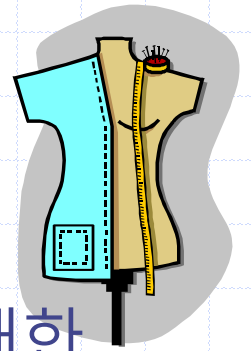
- $T_v$ 의  $v$ 에서  $w$ 로 향하는 경로는  $i$ 개의 간선을 가진다
- $G_v$  내의  $v$ 에서  $w$ 로 향하는 모든 경로는 최소  $i$ 개의 간선을 가진다



# BFS 분석

- ◆ 정점과 간선의 **라벨**을 쓰고 읽는데 **O(1)** 시간 소요
  - 참고: 정점이나 간선을 구현하는 노드 위치의 기능성에 "*Visited*" 플래그를 포함하도록 확장 가능
- ◆ 각 정점은 두 번 라벨
  - 한 번은 *Fresh*로, 또 한 번은 *Visited*로
- ◆ 각 간선은 두 번 라벨
  - 한 번은 *Fresh*로, 또 한 번은 *Tree* 또는 *Cross*로
- ◆ 각 정점은 리스트  $L_i$ 에 한 번 삽입
- ◆ 메소드 **incidentEdges**는 각 정점에 대해 한 번 호출
- ◆ 그래프가 **인접리스트** 구조로 표현된 경우, **BFS**는 **O(n + m)** 시간에 수행
  - 참고:  $\sum_v \text{deg}(v) = 2m$

# BFS 템플릿 활용



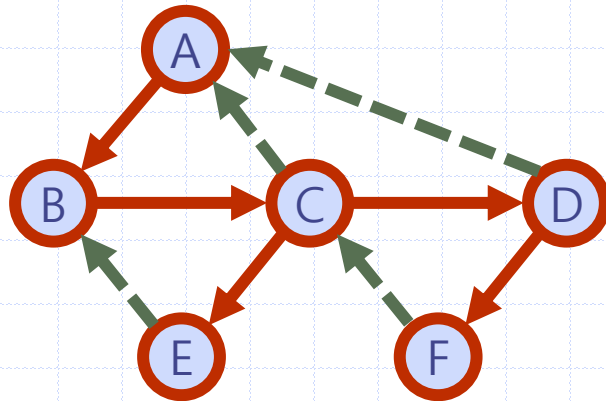
◆ 템플릿 메소드 패턴을 사용하여, 그래프  $G$ 에 대한 **BFS** 순회를 다음 문제들을  $O(n + m)$  시간에 해결하도록 **특화**할 수 있다

- $G$ 의 연결요소들을 계산하기
- $G$ 의 신장숲을 계산하기
- $G$  내의 단순 사이클 찾기 또는  $G$ 가 숲임을 보고하기
- $G$ 의 주어진 두 정점에 대해, 그 사이의 최소 간선으로 이루어진  $G$  내의 경로 찾기, 또는 그런 경로가 없음을 보고하기

# 비트리 간선

## 후향간선 ( $v, w$ )

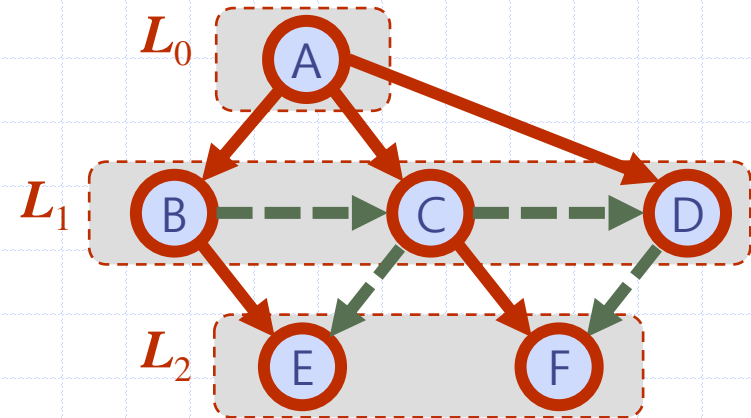
- 트리 간선들의 트리에서  $w$ 가  $v$ 의 조상



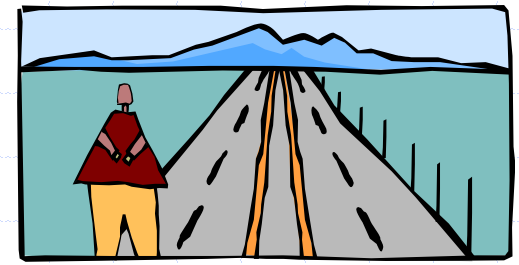
DFS

## 교차간선 ( $v, w$ )

- 트리 간선들의 트리에서  $w$ 가  $v$ 와 동일 또는 다음 레벨에 위치



BFS





# DFS와 BFS 응용

응용	DFS	BFS
신장숲 연결요소 경로 싸이클	√	√
최단경로		√
이중 연결요소	√	



# 응용문제: 경로 찾기

- ◆ 원형 메소드를 사용하여, 주어진 두 정점 사이의 경로를 찾기 위한 DFS의 특화를 의사코드로 작성하라
  - $\text{path}(G, v, z)$ :  $G$ 의 주어진 두 정점  $v$ 와  $z$  사이의 경로를 찾아 보고
- ◆ 힌트: 출발정점과 현재 정점 사이의 경로를 추적하기 위해 스택을 사용

# 해결: 경로 찾기

- ◆  $\text{path}(G, v, z)$ 는  $v$ 와  $z$ 를 각각 출발 및 도착점으로 하여  $\text{pathDFS}(G, v, z, S)$ 를 호출
- ◆  $v$ 와 현재 정점 사이의 경로를 추적하기 위해 스택  $S$ 를 사용
- ◆  $z$ 를 만나자마자  $S$ 에 누적된 내용을 경로로써 반환

Alg  $\text{path}(G, v, z)$

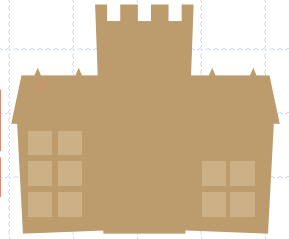
1.  $S \leftarrow \text{empty stack}$
2.  $\text{pathDFS}(G, v, z, S)$
3. return  $S.\text{elements}()$

Alg  $\text{pathDFS}(G, v, z, S)$

1.  $l(v) \leftarrow \text{Visited}$
2.  $S.\text{push}(v)$
3. if ( $v = z$ )  
    return
4. for each  $e \in G.\text{incidentEdges}(v)$   
    if ( $l(e) = \text{Fresh}$ )  
         $w \leftarrow \text{opposite}(v, e)$   
        if ( $l(w) = \text{Fresh}$ )  
             $l(e) \leftarrow \text{Tree}$   
             $S.\text{push}(e)$   
             $\text{pathDFS}(G, w, z, S)$   
             $S.\text{pop}()$  {  $e$  gets popped }  
        else  
             $l(e) \leftarrow \text{Back}$
5.  $S.\text{pop}()$  {  $v$  gets popped }

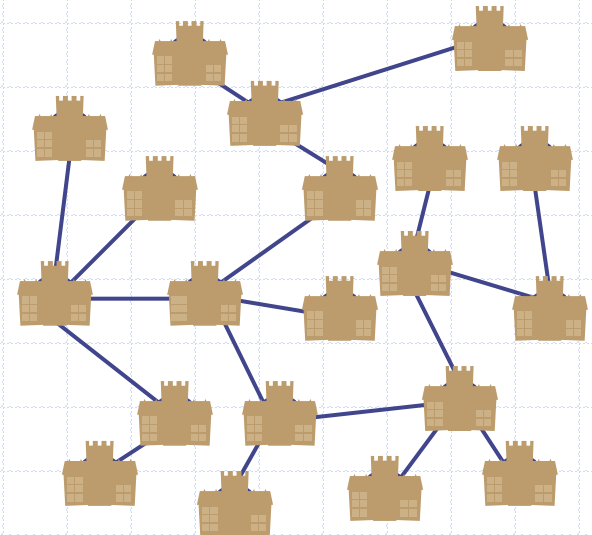


# 응용문제: 자유트리의 중심



- ◆ S 대학교와 전세계의 많은 대학들은 멀티미디어에 관한 협력과제를 수행하고 있다
- ◆ **자유트리**를 형성하는 통신선로를 사용하여 이 대학들을 연결하기 위한 컴퓨터 네트워크를 구축한다
- ◆ 대학들은 모든 대학 간에 데이터를 공유하기 위해 대학들 가운데 한 곳에 **파일서버**를 설치하기로 결정하였다
- ◆ 통신선로의 전송시간은 선로 구성과 동기화에 지배되므로, 데이터 전송의 비용은 사용된 **선로의 수**에 비례한다
- ◆ 그러므로, 파일서버를 **중심** 위치에 놓는 것이 바람직하다

자유트리  $T$

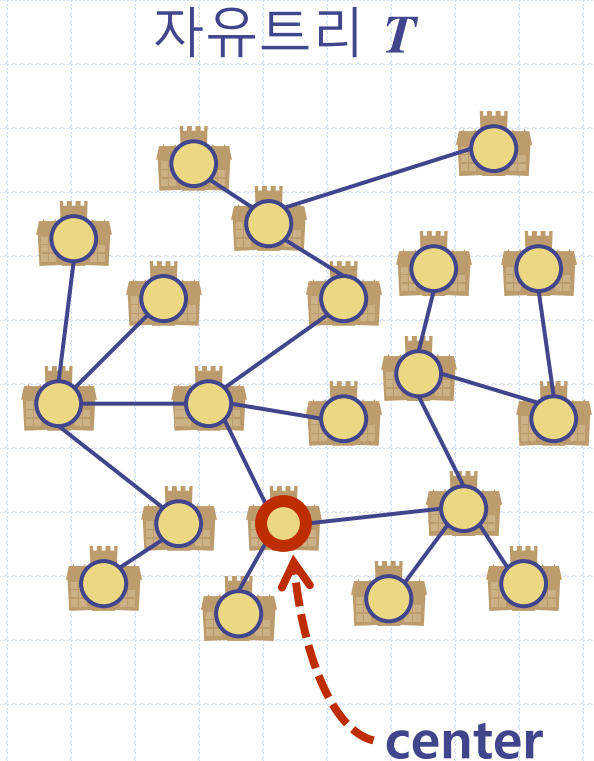


# 응용 문제: 자유트리의 중심

## (conti.)

- ◆ 자유트리  $T$ 와  $T$ 의 한 노드  $v$ 가 주어졌을 때,  $v$ 의 **이심률**(eccentricity)이란  $v$ 로부터  $T$ 의 다른 노드로의 경로 가운데 최장경로의 길이를 말한다
- ◆ 최소의 이심률을 가지는  $T$ 의 노드를  $T$ 의 **중심**(center)이라 부른다

- 주어진  $n$ -노드 자유트리  $T$ 에 대해,  $T$ 의 중심을 찾는 효율적인 알고리즘을 **의사코드**로 설계하라
- 중심은 **유일**한가? 아니라면, 자유tree는 중심을 몇 개까지 가질 수 있는가?



# 해결: 개요

## ◆ 관찰

- 자유트리  $T$ 의 중심은 트리의 모든 잎들을 삭제하더라도 변하지 않는다(반대로, 모든 잎에 자식들을 추가하더라도 마찬가지)
- 최소 세 개 노드의 트리에서 잎은 중심이 될 수 없다

◆ 그러므로, 주어진 자유트리의 잎을 계속 삭제하여 하나 또는 두 개의 노드만 남게 되면 이것이 트리의 중심이다

## ◆ 잎의 삭제

- 원래 트리의 복사본에서 잎을 실제로 삭제할 수도, 혹은 잎에 표시함으로써 모의적으로 삭제 가능
- 어떤 "삭제" 작업이던 트리를 순회하며 (임의의 노드에서 출발) 잎을 삭제하거나 표시함으로써 수행 가능
- 최악의 경우, 삭제 작업은  $O(n)$  시간 소요 - 즉, 전체적으로는 2차 시간에 수행(편향트리를 생각해볼 것)

# 해결: 개요 (conti.)

- ◆ 결과 중심 그래프  $G'$ 가 단 한 개의 노드라면, 이것이  $G$ 의
- ◆  $G'$ 가 두 개의 노드라면, 이들 가운데 아무거나  $G$ 의
- ◆ 알고리즘은  $G$ 의 중심만을 반환하지 이심율을 반환하지는 않는다 – 하지만 이는 잎 삭제 작업의 회수를 저장한다면 쉽게 구할 수 있다

# 해결: 알고리즘 (conti.)

**Alg** *findCenter*( $G$ )

**input** a free tree  $G$

**output** a center of  $G$

1.  $G' \leftarrow$  a copy of  $G$
2. **while** ( $G'.\text{numVertices}() > 2$ )  
     $\text{removeLeaves}(G', G'.\text{aVertex}(), \text{Null})$
3. **return**  $G'.\text{aVertex}()$

**Alg** *removeLeaves*( $G, v, p$ )

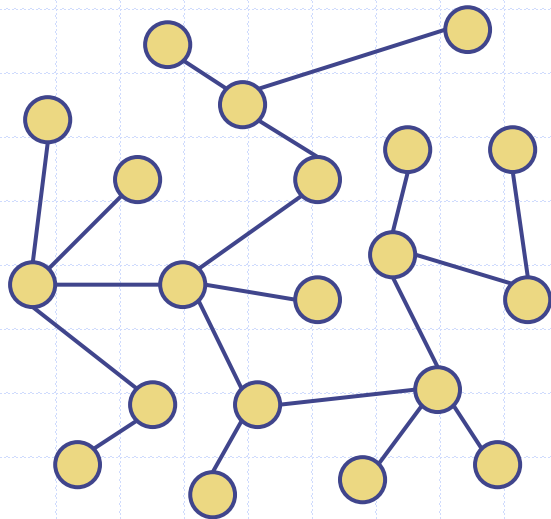
**input** a free tree  $G$ , a vertex  $v$ ,  
parent vertex  $p$  of  $v$  in the  
tree traversal

**output** a free tree  $G$  with its  
leaves removed

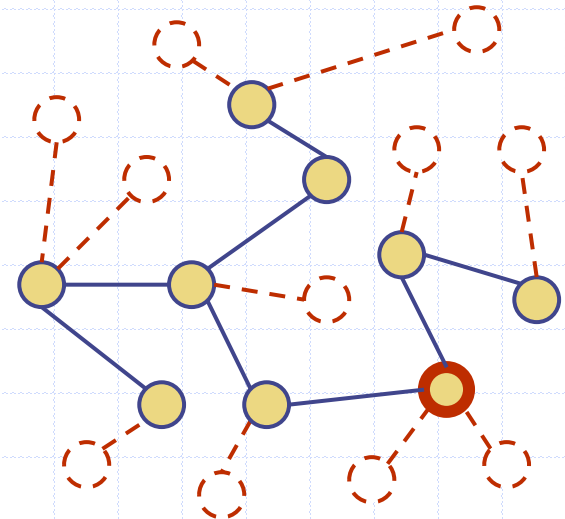
1.  $c \leftarrow 0$
2. **for each**  $e \in G.\text{incidentEdges}(v)$   
     $c \leftarrow c + 1$   
     $w \leftarrow G.\text{opposite}(v, e)$   
    **if** ( $w \neq p$ )  
         $\text{removeLeaves}(G, w, v)$
3. **if** ( $c = 1$ )  
     $G.\text{removeVertex}(v)$

# 해결: 수행 예 (conti.)

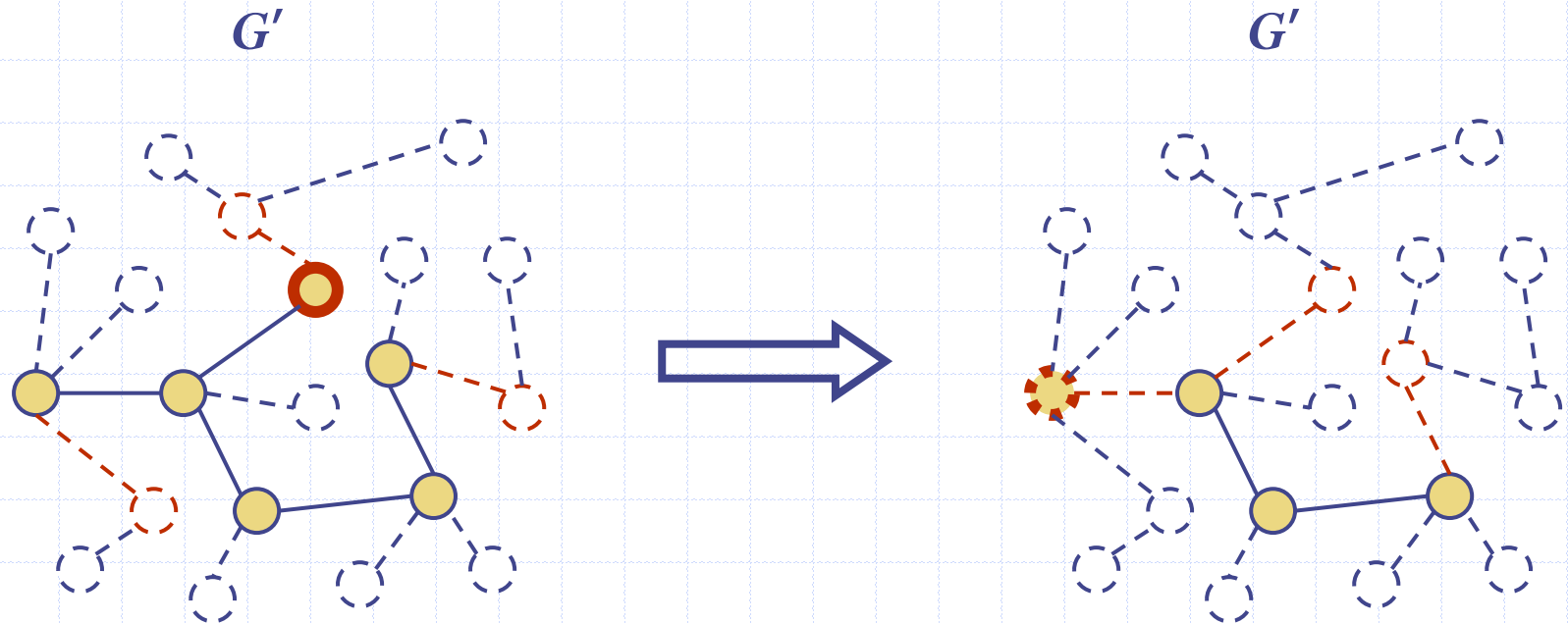
예:  $G = G'$



$G'$



# 해결: 수행 예 (conti.)



# 해결: 수행 예 (conti.)

