

Graphs

Definition, Implementation and Traversal

Graphs

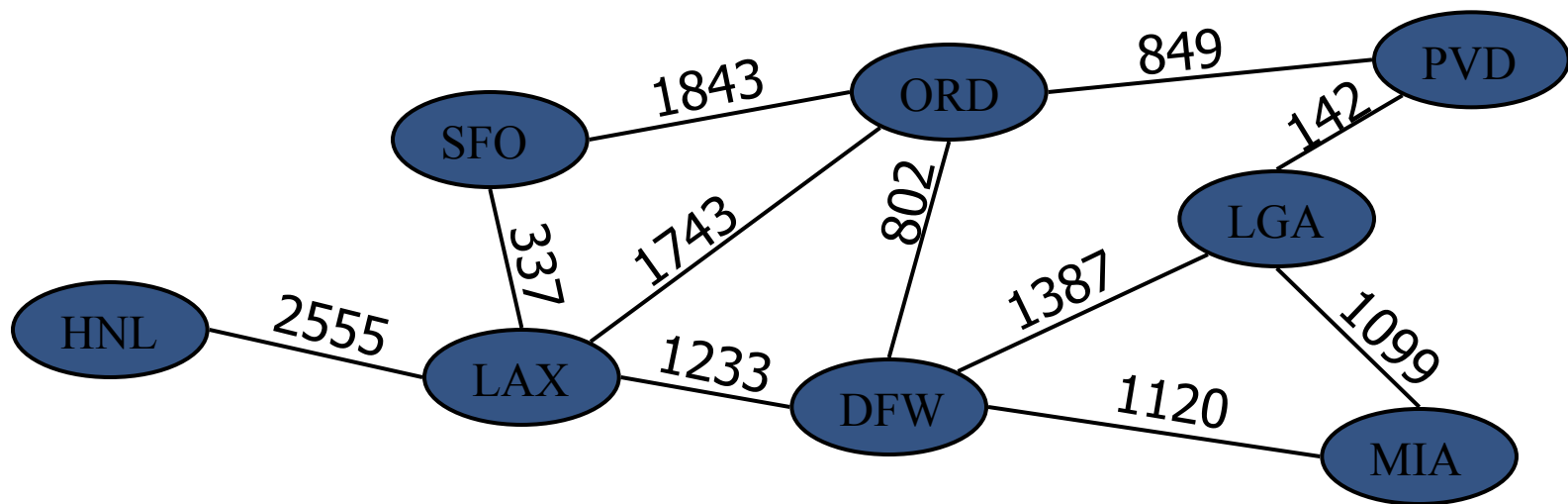
- Formally speaking, a graph is a pair (V, E) , where
 - V is a set of nodes, called vertices
 - E is a collection of pairs of vertices, called edges
 - Vertices and edges are positions and store elements



Graphs

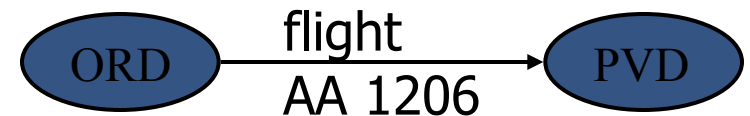
■ Example:

- A vertex represents an airport and stores the three-letter airport code
- An edge represents a flight route between two airports and stores the mileage of the route



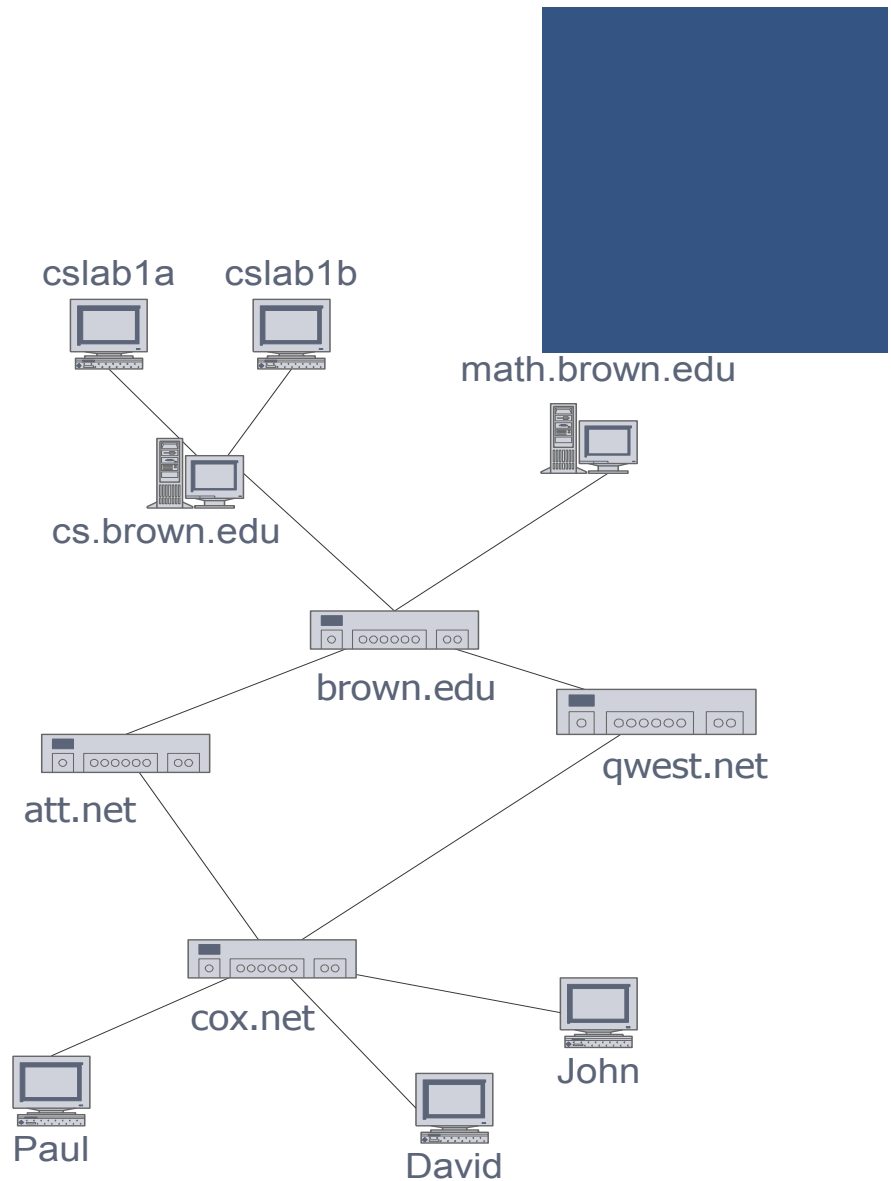
Edge Types

- 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



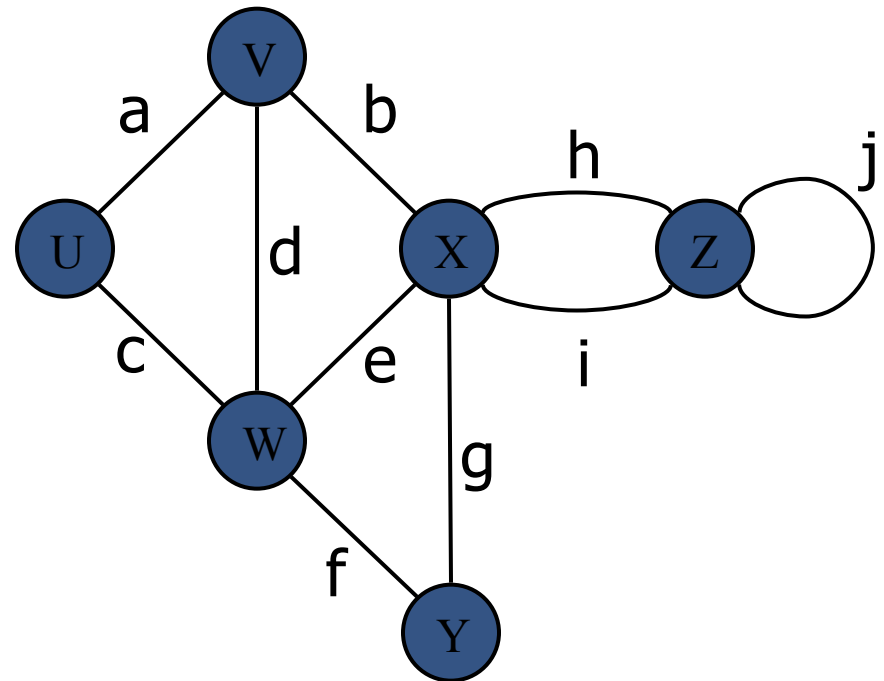
Applications

- Electronic circuits
 - Printed circuit board
 - Integrated circuit
- Transportation networks
 - Highway network
 - Flight network
- Computer networks
 - Local area network
 - Internet
 - Web
- Databases
 - Entity-relationship diagram



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



Terminology (cont.)

■ Path

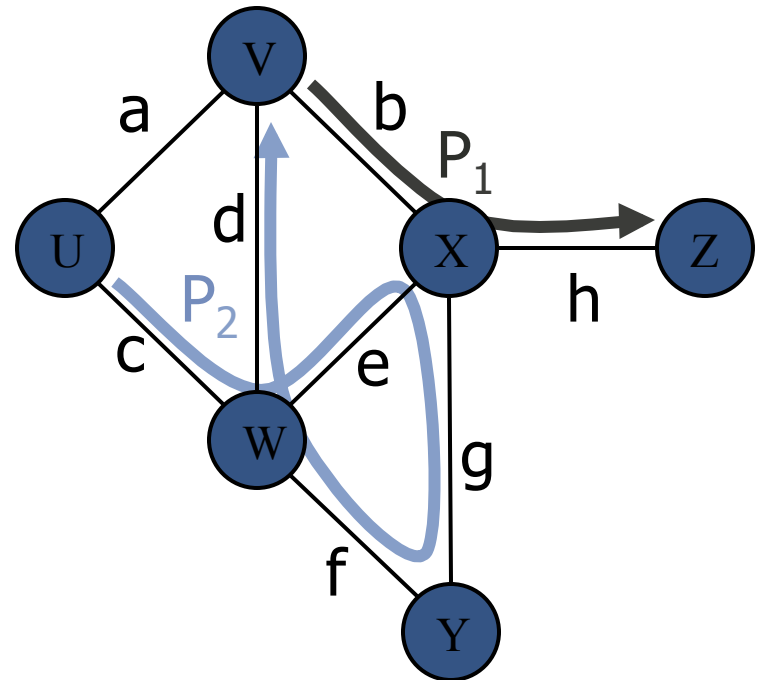
- sequence of alternating vertices and edges
- begins with a vertex
- ends with a vertex
- each edge is preceded and followed by its endpoints

■ Simple path

- path such that all its vertices and edges are distinct

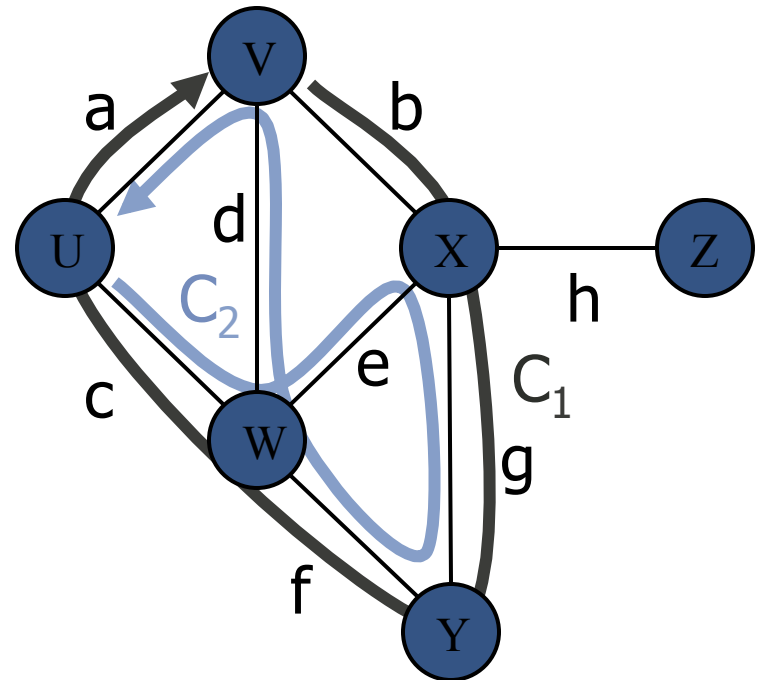
■ Examples

- $P_1 = (V, b, X, h, Z)$ is a simple path
- $P_2 = (U, c, W, e, X, g, Y, f, W, d, V)$ is a path that is not simple



Terminology (cont.)

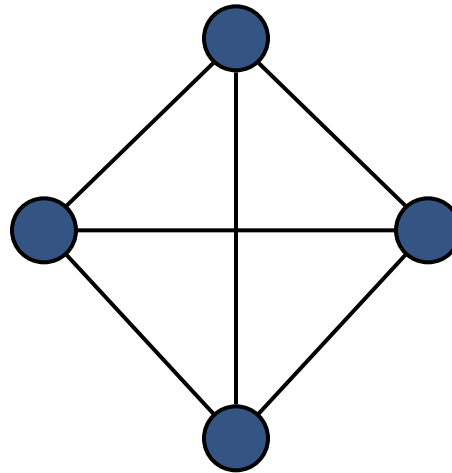
- Cycle
 - circular sequence of alternating vertices and edges
 - each edge is preceded and followed by its endpoints
- Simple cycle
 - cycle such that all its vertices and edges are distinct
- Examples
 - $C_1 = (V, b, X, g, Y, f, W, c, U, a, \hookrightarrow)$ is a simple cycle
 - $C_2 = (U, c, W, e, X, g, Y, f, W, d, V, a, \hookrightarrow)$ is a cycle that is not simple



Properties

Notation

n	number of vertices
m	number of edges
$\deg(v)$	degree of vertex v



Example

- $n = 4$
- $m = 6$
- $\deg(v) = 3$

Properties

- Property 1
 - $\sum_v \deg(v) = 2m$
 - Proof: each edge is counted twice
- Property 2
 - In an undirected graph with no self-loops and no multiple edges
 - $m \leq n(n-1)/2$
 - Proof: each vertex has degree at most $(n-1)$
- What is the bound for a directed graph?

Main Methods of 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`

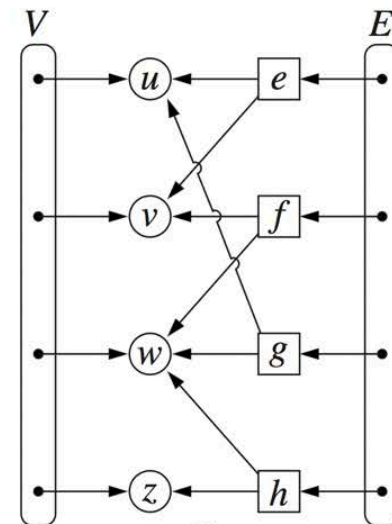
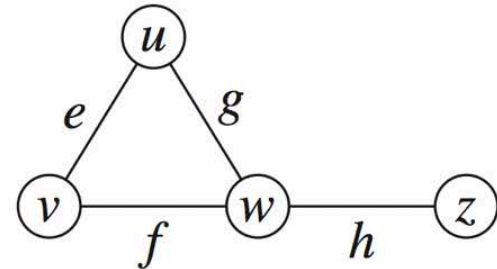
Main Methods of the Graph ADT



- 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`
- Iterable collection methods
 - `incidentEdges(v)`: edges incident to `v`
 - `vertices()`: all vertices in the graph
 - `edges()`: all edges in the graph

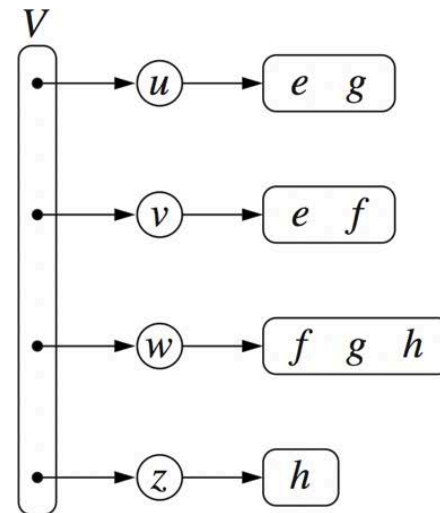
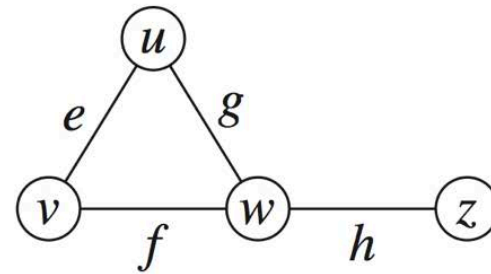
Edge List Structure

- Vertex object
 - element
 - reference to position in vertex sequence
- Edge object
 - element
 - origin vertex object
 - destination vertex object
 - reference to position in edge sequence
- Vertex sequence
 - sequence of vertex objects
- Edge sequence
 - sequence of edge objects



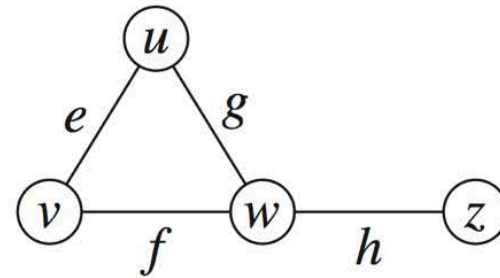
Adjacency List Structure

- Incidence sequence for each vertex
 - sequence of references to edge objects of incident edges
- Augmented edge objects
 - references to associated positions in incidence sequences of end vertices



Adjacency Matrix Structure

- Edge list structure
- Augmented vertex objects
 - Integer key (index) associated with vertex
- 2D-array adjacency array
 - Reference to edge object for adjacent vertices
 - Null for non adjacent vertices
- The “old fashioned” version just has 0 for no edge and 1 for edge



		0	1	2	3	
u	\longrightarrow	0		e	g	
v	\longrightarrow	1	e		f	
w	\longrightarrow	2	g	f		h
z	\longrightarrow	3			h	

Summary on the representations

- In each representation, we maintain a collection to store the vertices of a graph. However, the representations differ greatly in the way they organize the edges.

Summary on the representations

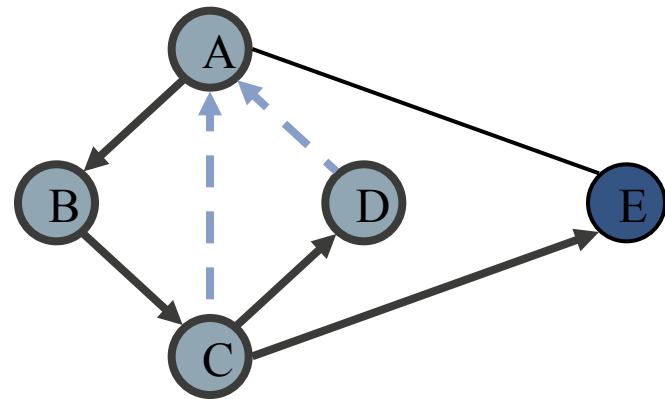
- In an **edge list**, we maintain an unordered list of all edges. This minimally suffices, but there is no efficient way to locate a particular edge (u,v) , or the set of all edges incident to a vertex v .
- In an **adjacency list**, we additionally maintain, for each vertex, a separate list containing those edges that are incident to the vertex. This organization allows us to more efficiently find all edges incident to a given vertex.
- An **adjacency matrix** provides worst-case $O(1)$ access to a specific edge (u,v) by maintaining an $n \times n$ matrix, for a graph with n vertices. Each slot is dedicated to storing a reference to the edge (u,v) for a particular pair of vertices u and v ; if no such edge exists, the slot will store null.

Performance

<ul style="list-style-type: none"> ▪ n vertices, m edges ▪ no parallel edges ▪ no self-loops 	Edge List	Adjacency List	Adjacency Matrix
Space	$n + m$	$n + m$	n^2
incidentEdges(v)	m	deg(v)	n
areAdjacent (v, w)	m	min(deg(v), deg(w))	1
insertVertex(o)	1	1	n^2
insertEdge(v, w, o)	1	1	1
removeVertex(v)	m	deg(v)	n^2
removeEdge(e)	1	1	1

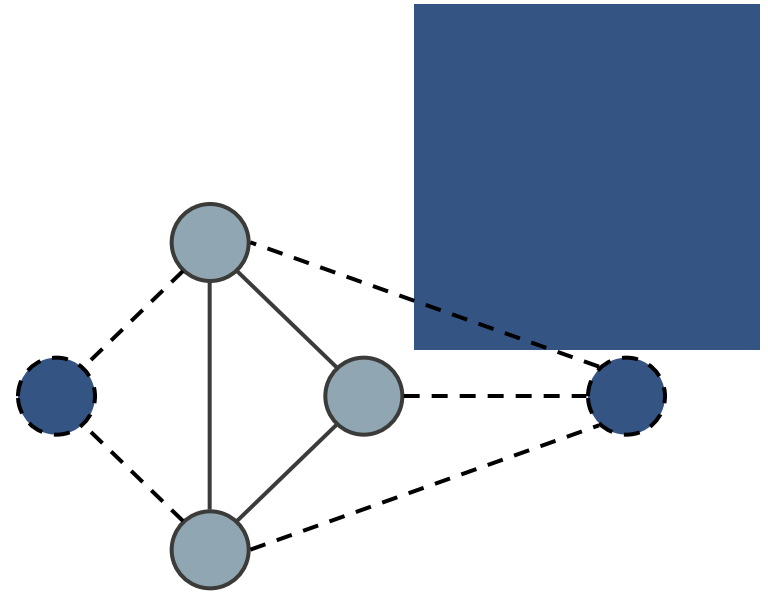
Graph Traversal

- How to visit all vertices?

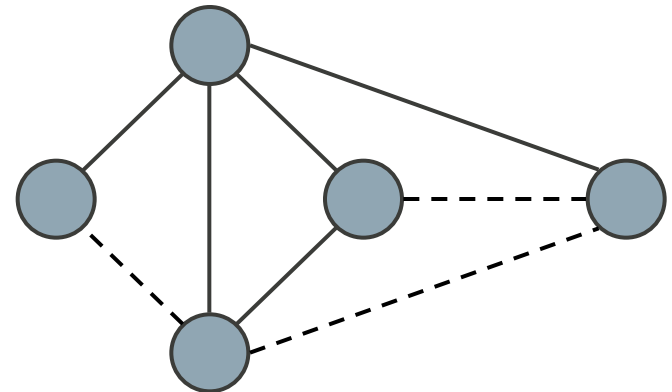


Subgraphs

- 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



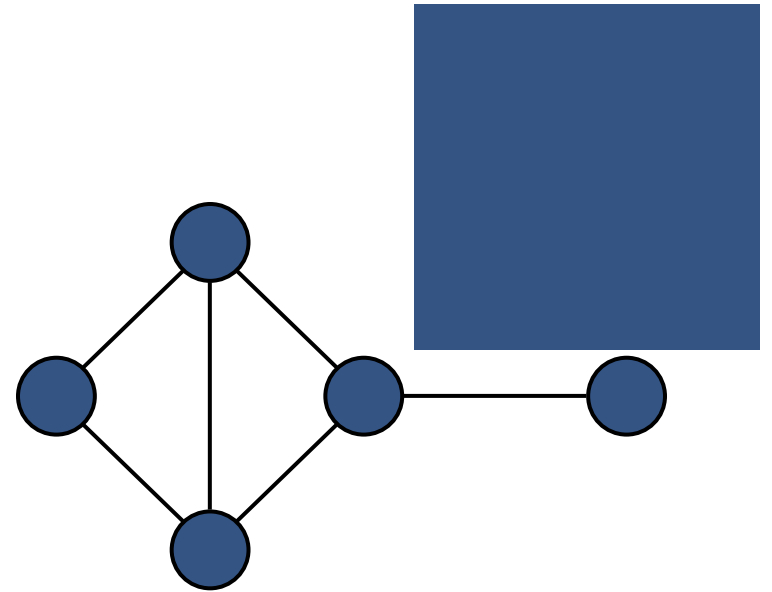
Subgraph



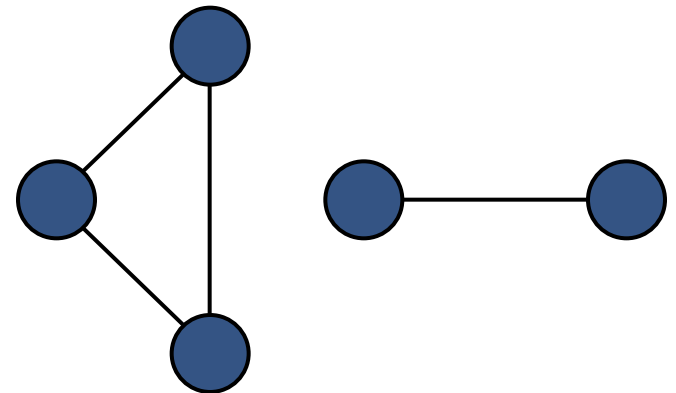
Spanning subgraph

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



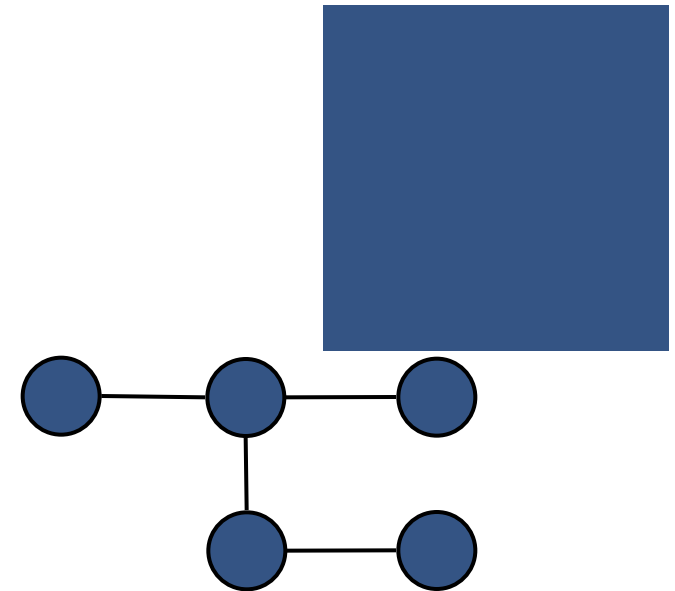
Non connected graph with two connected components

Trees and Forests

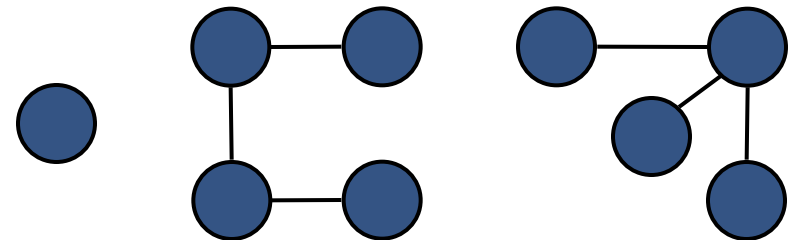
- A (free) tree is an undirected graph T such that
 - T is connected
 - T has no cycles

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



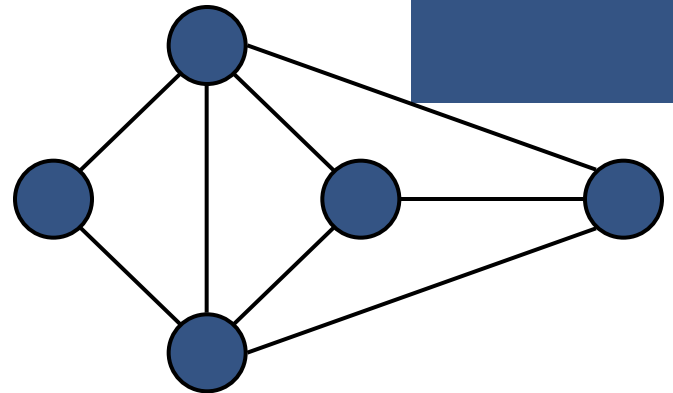
Tree



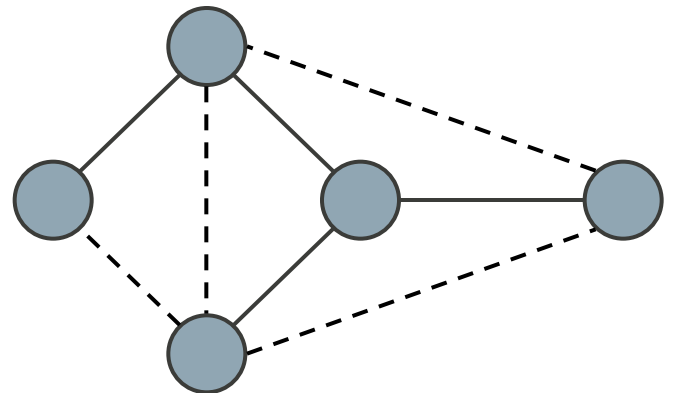
Forest

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
 - Determines whether G is connected
 - Computes the connected components of G
 - Computes a spanning forest of G
- DFS on a graph with n vertices and m edges takes $O(n + m)$ time
- 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
- Depth-first search is to graphs what Euler tour is to binary trees

DFS Algorithm from a Vertex

Algorithm DFS(G, u):

Input: A graph G and a vertex u of G

Output: A collection of vertices reachable from u , with their discovery edges

Mark vertex u as visited.

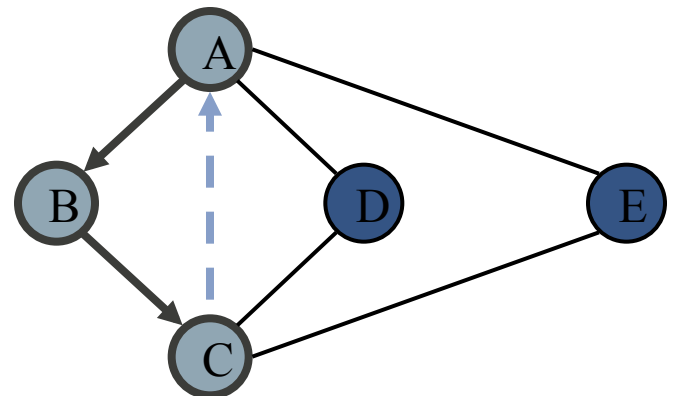
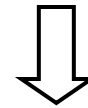
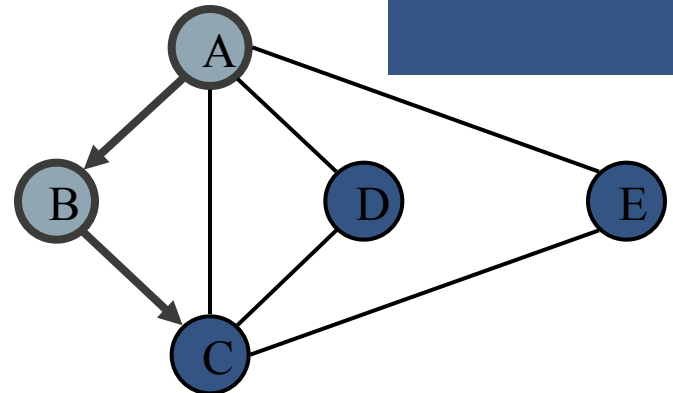
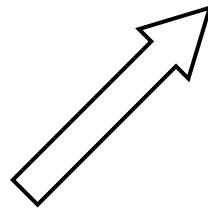
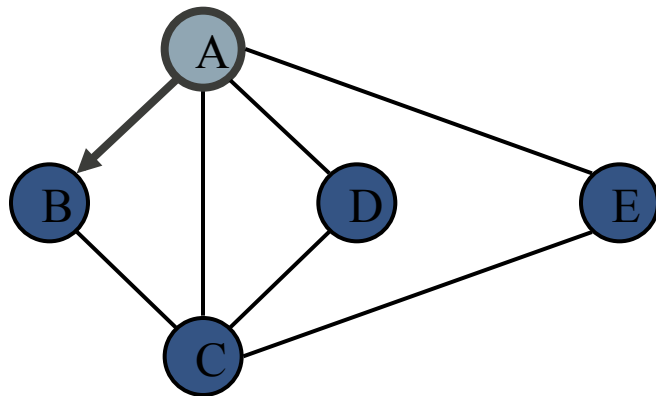
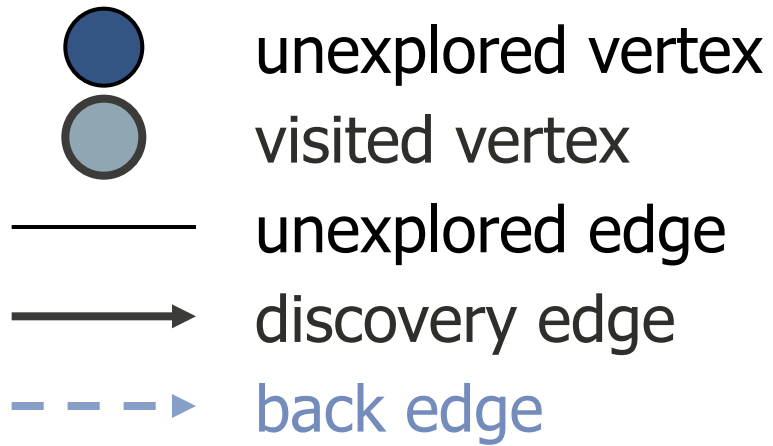
for each of u 's outgoing edges, $e = (u, v)$ **do**

if vertex v has not been visited **then**

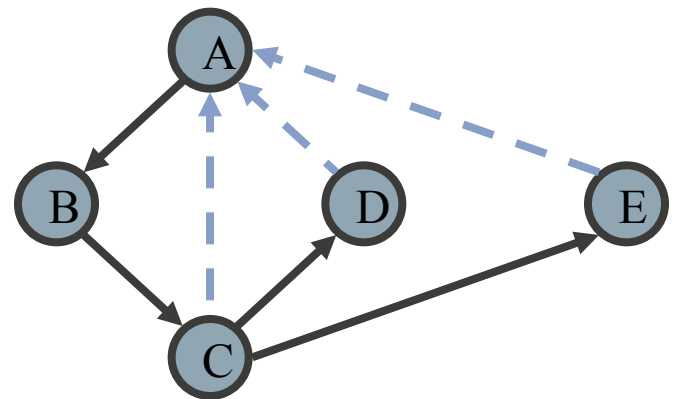
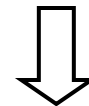
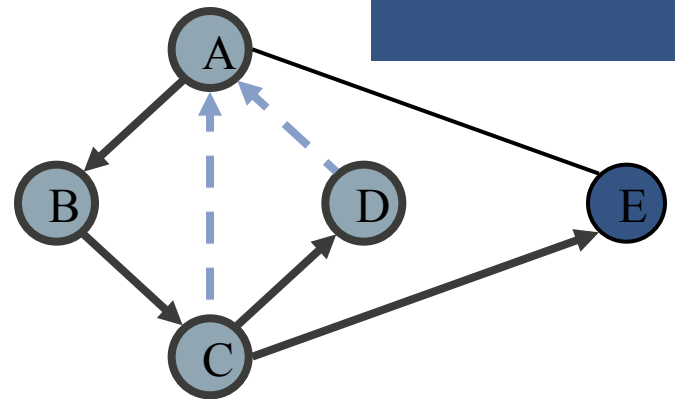
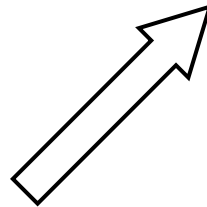
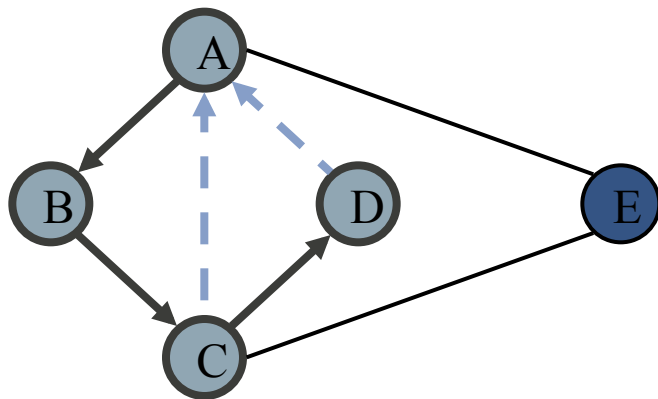
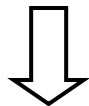
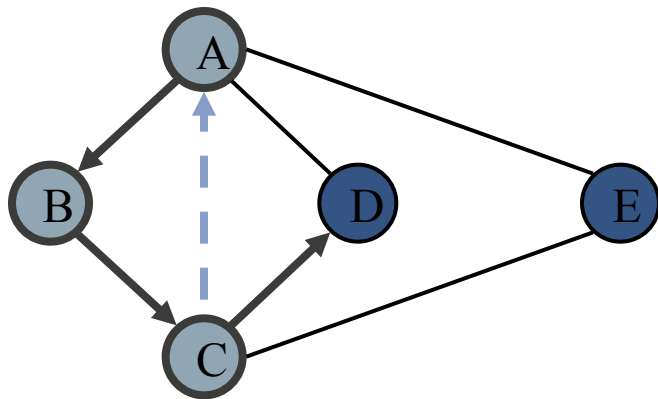
 Record edge e as the discovery edge for vertex v .

 Recursively call DFS(G, v).

Example

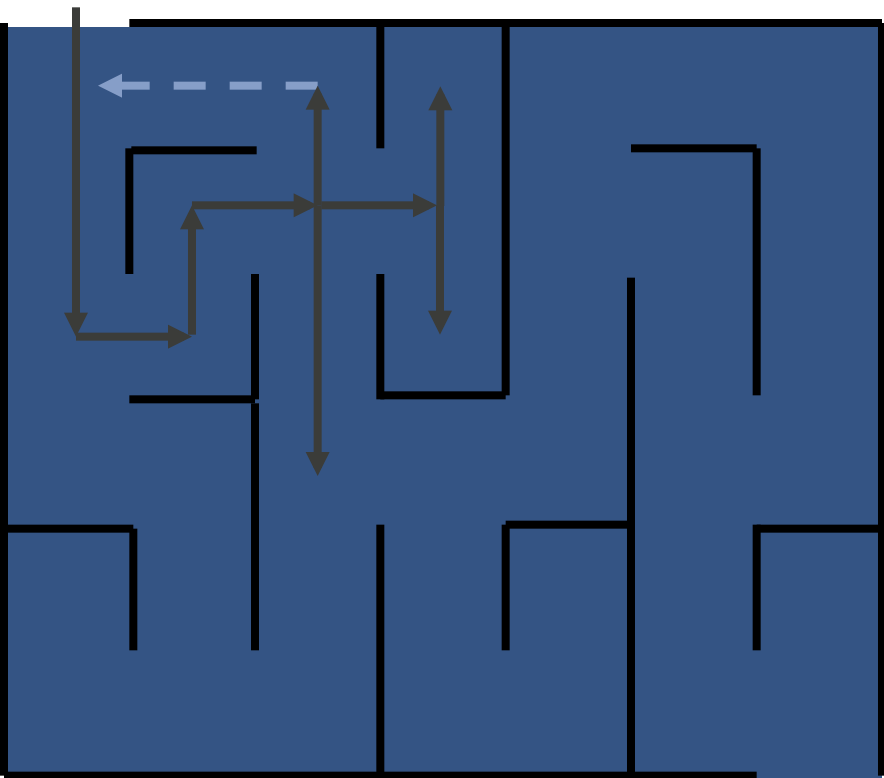


Example (cont.)



the 1990s, the number of people in the United States who are 65 years of age or older has increased by 50 percent, and the number of people 75 years of age or older has increased by 100 percent. The number of people 85 years of age or older has increased by 200 percent. The number of people 95 years of age or older has increased by 400 percent. The number of people 100 years of age or older has increased by 1,000 percent. The number of people 105 years of age or older has increased by 2,000 percent. The number of people 110 years of age or older has increased by 4,000 percent. The number of people 115 years of age or older has increased by 8,000 percent. The number of people 120 years of age or older has increased by 16,000 percent. The number of people 125 years of age or older has increased by 32,000 percent. The number of people 130 years of age or older has increased by 64,000 percent. The number of people 135 years of age or older has increased by 128,000 percent. The number of people 140 years of age or older has increased by 256,000 percent. The number of people 145 years of age or older has increased by 512,000 percent. The number of people 150 years of age or older has increased by 1,024,000 percent. The number of people 155 years of age or older has increased by 2,048,000 percent. The number of people 160 years of age or older has increased by 4,096,000 percent. The number of people 165 years of age or older has increased by 8,192,000 percent. The number of people 170 years of age or older has increased by 16,384,000 percent. The number of people 175 years of age or older has increased by 32,768,000 percent. The number of people 180 years of age or older has increased by 65,536,000 percent. The number of people 185 years of age or older has increased by 131,072,000 percent. The number of people 190 years of age or older has increased by 262,144,000 percent. The number of people 195 years of age or older has increased by 524,288,000 percent. The number of people 200 years of age or older has increased by 1,048,576,000 percent. The number of people 205 years of age or older has increased by 2,097,152,000 percent. The number of people 210 years of age or older has increased by 4,194,304,000 percent. The number of people 215 years of age or older has increased by 8,388,608,000 percent. The number of people 220 years of age or older has increased by 16,777,216,000 percent. The number of people 225 years of age or older has increased by 33,554,432,000 percent. The number of people 230 years of age or older has increased by 67,108,864,000 percent. The number of people 235 years of age or older has increased by 134,217,728,000 percent. The number of people 240 years of age or older has increased by 268,435,456,000 percent. The number of people 245 years of age or older has increased by 536,870,912,000 percent. The number of people 250 years of age or older has increased by 1,073,741,824,000 percent. The number of people 255 years of age or older has increased by 2,147,483,648,000 percent. The number of people 260 years of age or older has increased by 4,294,967,296,000 percent. The number of people 265 years of age or older has increased by 8,589,934,592,000 percent. The number of people 270 years of age or older has increased by 17,179,869,184,000 percent. The number of people 275 years of age or older has increased by 34,359,738,368,000 percent. The number of people 280 years of age or older has increased by 68,719,476,736,000 percent. The number of people 285 years of age or older has increased by 137,438,953,472,000 percent. The number of people 290 years of age or older has increased by 274,877,906,944,000 percent. The number of people 295 years of age or older has increased by 549,755,813,888,000 percent. The number of people 300 years of age or older has increased by 1,099,511,627,776,000 percent. The number of people 305 years of age or older has increased by 2,199,023,255,552,000 percent. The number of people 310 years of age or older has increased by 4,398,046,511,104,000 percent. The number of people 315 years of age or older has increased by 8,796,093,022,208,000 percent. The number of people 320 years of age or older has increased by 17,592,186,044,416,000 percent. The number of people 325 years of age or older has increased by 35,184,372,088,832,000 percent. The number of people 330 years of age or older has increased by 70,368,744,177,664,000 percent. The number of people 335 years of age or older has increased by 140,737,488,355,328,000 percent. The number of people 340 years of age or older has increased by 281,474,976,710,656,000 percent. The number of people 345 years of age or older has increased by 562,949,953,421,312,000 percent. The number of people 350 years of age or older has increased by 1,125,899,906,842,624,000 percent. The number of people 355 years of age or older has increased by 2,251,799,813,685,248,000 percent. The number of people 360 years of age or older has increased by 4,503,599,627,370,496,000 percent. The number of people 365 years of age or older has increased by 9,007,199,254,740,992,000 percent. The number of people 370 years of age or older has increased by 18,014,398,509,481,984,000 percent. The number of people 375 years of age or older has increased by 36,028,797,018,963,968,000 percent. The number of people 380 years of age or older has increased by 72,057,594,037,927,936,000 percent. The number of people 385 years of age or older has increased by 144,115,188,075,855,872,000 percent. The number of people 390 years of age or older has increased by 288,230,376,151,711,744,000 percent. The number of people 395 years of age or older has increased by 576,460,752,303,423,488,000 percent. The number of people 400 years of age or older has increased by 1,152,921,504,606,846,976,000 percent. The number of people 405 years of age or older has increased by 2,305,843,009,213,693,952,000 percent. The number of people 410 years of age or older has increased by 4,611,686,018,427,387,904,000 percent. The number of people 415 years of age or older has increased by 9,223,372,036,854,775,808,000 percent. The number of people 420 years of age or older has increased by 18,446,744,073,709,551,616,000 percent. The number of people 425 years of age or older has increased by 36,893,488,147,419,103,232,000 percent. The number of people 430 years of age or older has increased by 73,786,976,294,838,206,464,000 percent. The number of people 435 years of age or older has increased by 147,573,952,589,676,412,928,000 percent. The number of people 440 years of age or older has increased by 295,147,905,179,352,825,856,000 percent. The number of people 445 years of age or older has increased by 590,295,810,358,705,651,712,000 percent. The number of people 450 years of age or older has increased by 1,180,591,620,717,411,303,424,000 percent. The number of people 455 years of age or older has increased by 2,361,183,241,434,822,606,848,000 percent. The number of people 460 years of age or older has increased by 4,722,366,482,869,645,213,696,000 percent. The number of people 465 years of age or older has increased by 9,444,732,965,739,290,427,392,000 percent. The number of people 470 years of age or older has increased by 18,889,465,931,478,580,854,784,000 percent. The number of people 475 years of age or older has increased by 37,778,931,862,957,161,709,568,000 percent. The number of people 480 years of age or older has increased by 75,557,863,725,914,323,419,136,000 percent. The number of people 485 years of age or older has increased by 151,115,727,451,828,646,838,272,000 percent. The number of people 490 years of age or older has increased by 302,231,454,903,657,293,676,544,000 percent. The number of people 495 years of age or older has increased by 604,462,909,807,314,587,353,088,000 percent. The number of people 500 years of age or older has increased by 1,208,925,819,614,629,174,706,176,000 percent. The number of people 505 years of age or older has increased by 2,417,851,639,229,258,349,412,352,000 percent. The number of people 510 years of age or older has increased by 4,835,703,278,458,516,698,824,704,000 percent. The number of people 515 years of age or older has increased by 9,671,406,556,917,033,397,649,408,000 percent. The number of people 520 years of age or older has increased by 19,342,813,113,834,066,795,298,816,000 percent. The number of people 525 years of age or older has increased by 38,685,626,227,668,133,590,597,632,000 percent. The number of people 530 years of age or older has increased by 77,371,252,455,336,267,181,195,264,000 percent. The number of people 535 years of age or older has increased by 154,742,504,910,672,534,362,390,528,000 percent. The number of people 540 years of age or older has increased by 309,485,009,821,345,068,724,781,056,000 percent. The number of people 545 years of age or older has increased by 618,970,019,642,690,137,449,562,112,000 percent. The number of people 550 years of age or older has increased by 1,237,940,039,285,380,274,899,124,224,000 percent. The number of people 555 years of age or older has increased by 2,475,880,078,570,760,549,798,248,448,000 percent. The number of people 560 years of age or older has increased by 4,951,760,157,141,521,099,596,496,896,000 percent. The number of people 565 years of age or older has increased by 9,903,520,314,283,042,199,193,993,792,000 percent. The number of people 570 years of age or older has increased by 19,807,040,628,566,084,398,387,

-



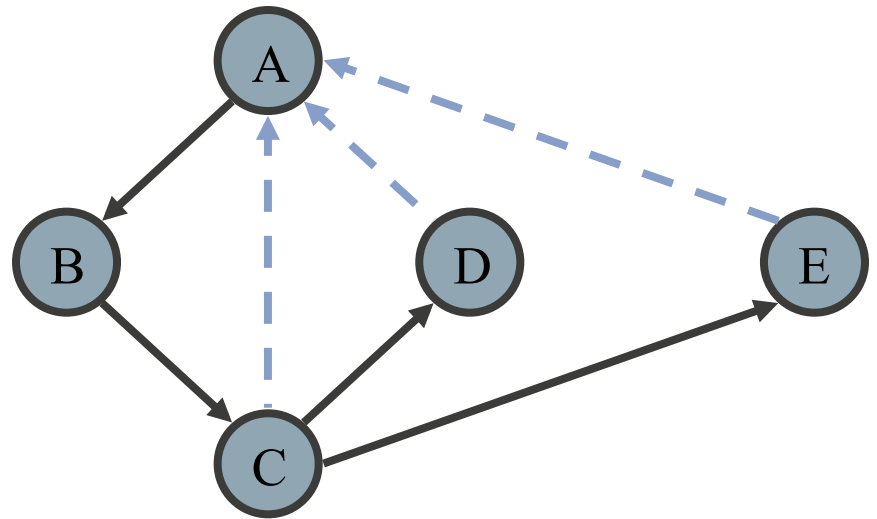
Properties of DFS

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



Analysis of DFS

- Setting/getting a vertex/edge label takes $O(1)$ time
- Each vertex is labeled twice
 - once as UNEXPLORED
 - once as VISITED
- Each edge is labeled twice
 - once 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
 - Recall that $\sum_v \deg(v) = 2m$

DFS Algorithm

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

Algorithm *DFS*(*G*)

Input graph *G*

Output labeling of the edges of *G*
as discovery edges and
back edges

```
for all  $u \in G.vertices()$ 
     $setLabel(u, UNEXPLORED)$ 
for all  $e \in G.edges()$ 
     $setLabel(e, UNEXPLORED)$ 
for all  $v \in G.vertices()$ 
    if  $getLabel(v) = UNEXPLORED$ 
         $DFS(G, v)$ 
```

Algorithm *DFS*(*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 discovery edges and back edges

```
 $setLabel(v, VISITED)$ 
for all  $e \in G.incidentEdges(v)$ 
    if  $getLabel(e) = UNEXPLORED$ 
         $w \leftarrow opposite(v, e)$ 
        if  $getLabel(w) = UNEXPLORED$ 
             $setLabel(e, DISCOVERY)$ 
             $DFS(G, w)$ 
        else
             $setLabel(e, BACK)$ 
```

Path Finding

- We can specialize the DFS algorithm to find a path between two given vertices u and z using the template method pattern
- We call $DFS(G, u)$ 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( $G, v, z$ )  
  setLabel( $v, VISITED$ )  
   $S.push(v)$   
  if  $v = z$   
    return  $S.elements()$   
  for all  $e \in G.incidentEdges(v)$   
    if getLabel( $e$ ) = UNEXPLORED  
       $w \leftarrow opposite(v, e)$   
      if getLabel( $w$ ) = UNEXPLORED  
        setLabel( $e, DISCOVERY$ )  
        S.push( $e$ )  
        pathDFS( $G, w, z$ )  
         $S.pop(e)$   
      else  
        setLabel( $e, BACK$ )  
   $S.pop(v)$ 
```

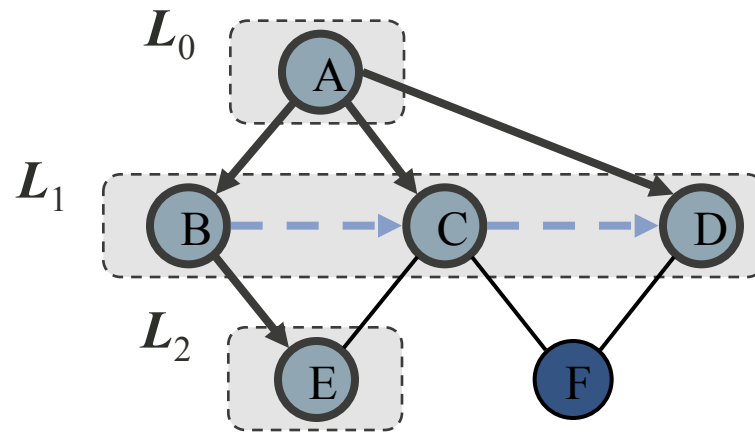

Cycle Finding

- We can specialize the DFS algorithm to find a simple cycle using the template method pattern
- We use a stack S to keep track of the path between the start vertex and the current vertex
- As soon as a back edge (v, w) is encountered, we return the cycle as the portion of the stack from the top to vertex w

```
Algorithm cycleDFS( $G, v, z$ )  
  setLabel( $v, VISITED$ )  
   $S.push(v)$   
  for all  $e \in G.incidentEdges(v)$   
    if getLabel( $e$ ) = UNEXPLORED  
       $w \leftarrow opposite(v, e)$   
       $S.push(e)$   
      if getLabel( $w$ ) = UNEXPLORED  
        setLabel( $e, DISCOVERY$ )  
        pathDFS( $G, w, z$ )  
         $S.pop(e)$   
      else  
         $T \leftarrow$  new empty stack  
        repeat  
           $o \leftarrow S.pop()$   
           $T.push(o)$   
        until  $o = w$   
        return  $T.elements()$   
   $S.pop(v)$ 
```

Breadth-First Search

- Traverse the graph level by level



Breadth-First Search



- Breadth-first search (BFS) is a general technique for traversing a graph
- A BFS traversal of a graph G
 - Visits all the vertices and edges of G
 - Determines whether G is connected
 - Computes the connected components of G
 - Computes a spanning forest of G
- BFS on a graph with n vertices and m edges takes $O(n + m)$ time
- BFS can be further extended to solve other graph problems
 - Find and report a path with the minimum number of edges between two given vertices
 - Find a simple cycle, if there is one

BFS Algorithm

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

Algorithm *BFS*(*G*)

Input graph *G*

Output labeling of the edges
and partition of the
vertices of *G*

```
for all u ∈ G.vertices()  
    setLabel(u, UNEXPLORED)  
for all e ∈ G.edges()  
    setLabel(e, UNEXPLORED)  
for all v ∈ G.vertices()  
    if getLabel(v) = UNEXPLORED  
        BFS(G, v)
```

Algorithm *BFS*(*G*, *s*)

*L*₀ ← new empty sequence

*L*₀.*addLast*(*s*)

setLabel(*s*, *VISITED*)

i ← 0

while ¬*L*_{*i*}.*isEmpty*()

*L*_{*i*+1} ← new empty sequence

for all *v* ∈ *L*_{*i*}.*elements*()

for all *e* ∈ *G.incidentEdges*(*v*)

if *getLabel*(*e*) = *UNEXPLORED*

w ← *opposite*(*v*, *e*)

if *getLabel*(*w*) = *UNEXPLORED*

setLabel(*e*, *DISCOVERY*)

setLabel(*w*, *VISITED*)

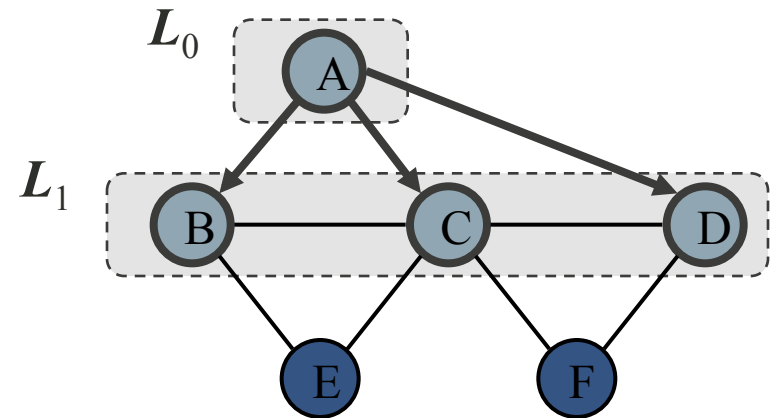
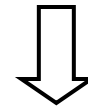
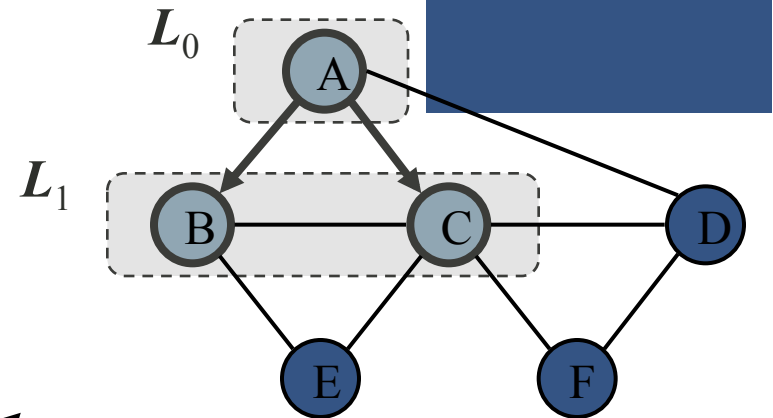
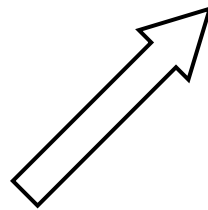
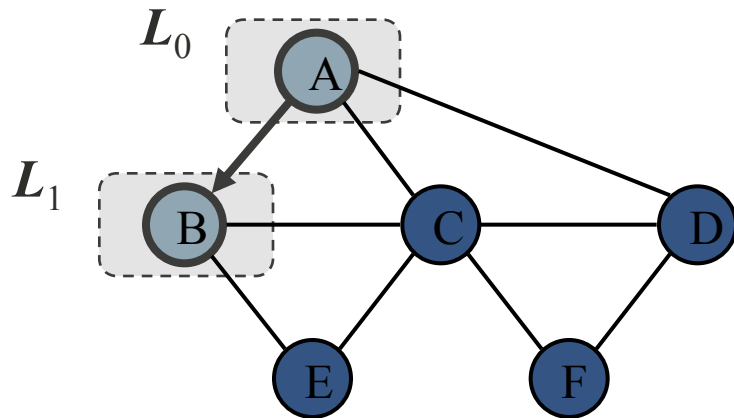
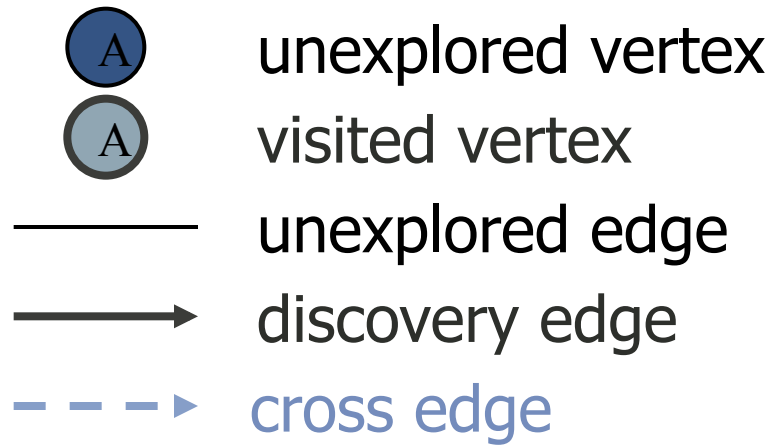
*L*_{*i*+1}.*addLast*(*w*)

else

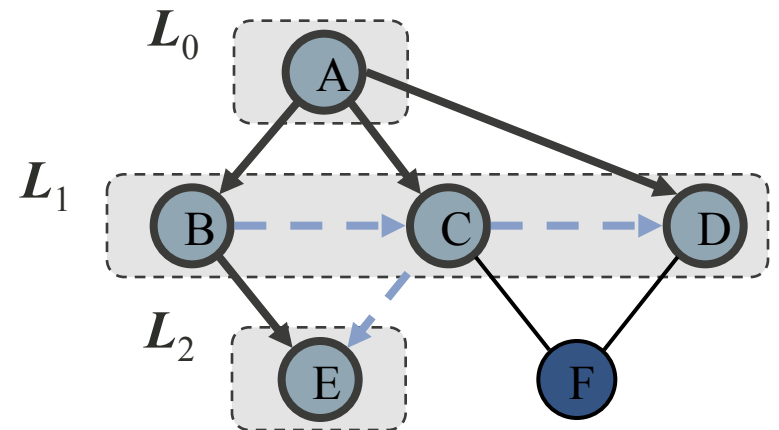
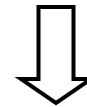
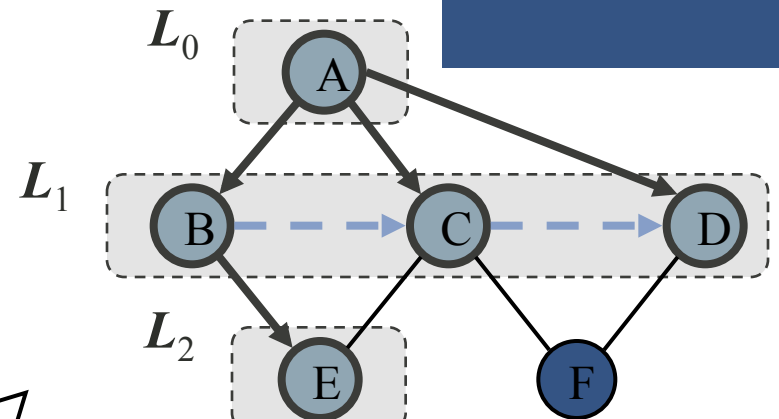
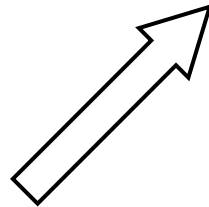
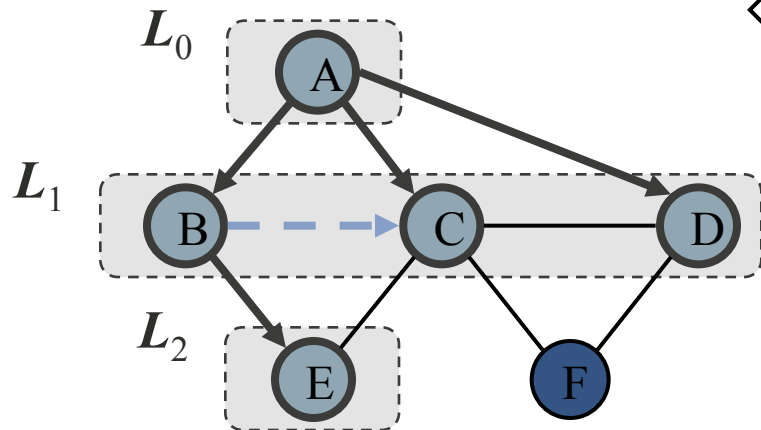
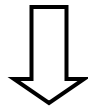
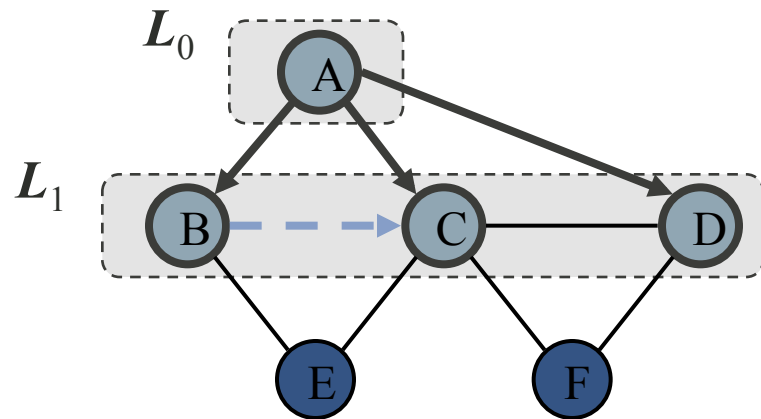
setLabel(*e*, *CROSS*)

i ← *i* + 1

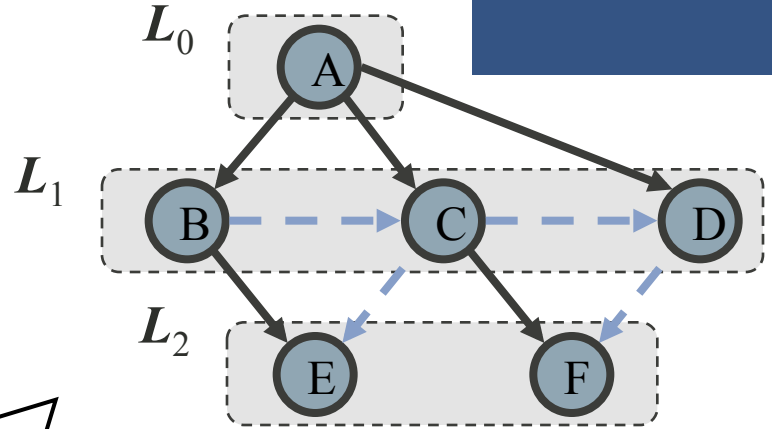
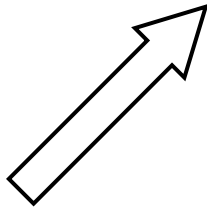
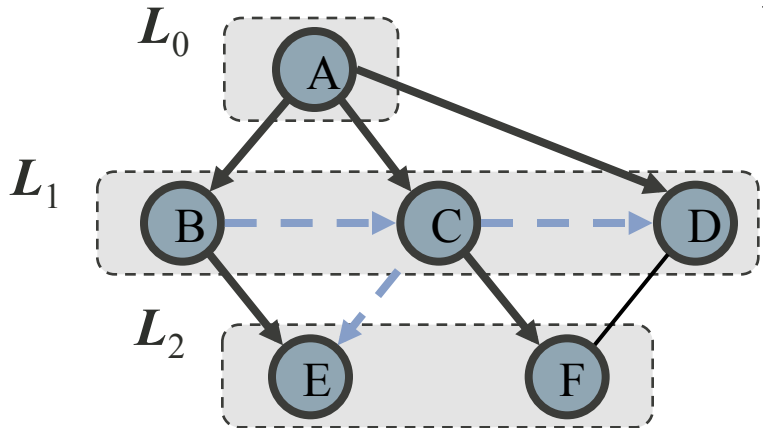
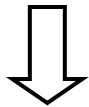
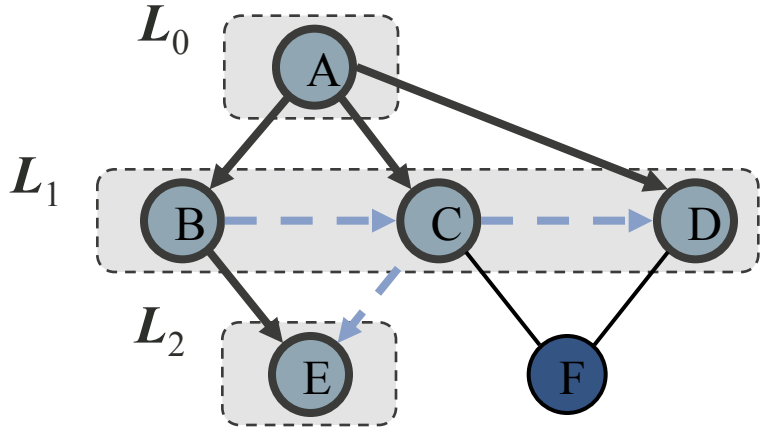
Example



Example (cont.)



Example (cont.)



Properties

Notation

G_s : connected component of s

Property 1

$BFS(G, s)$ visits all the vertices and edges of G_s

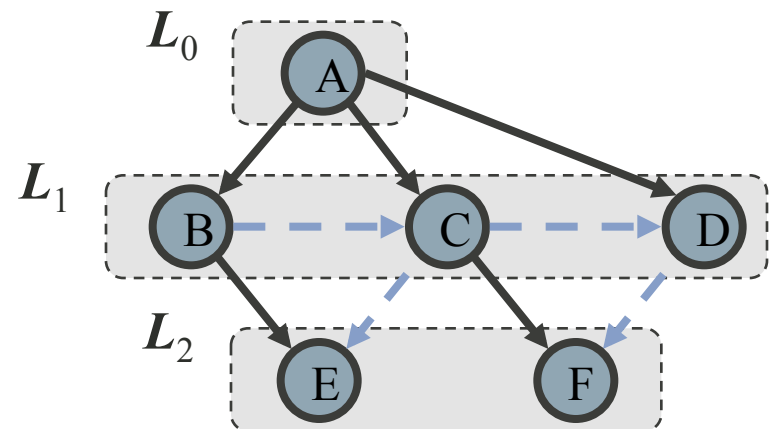
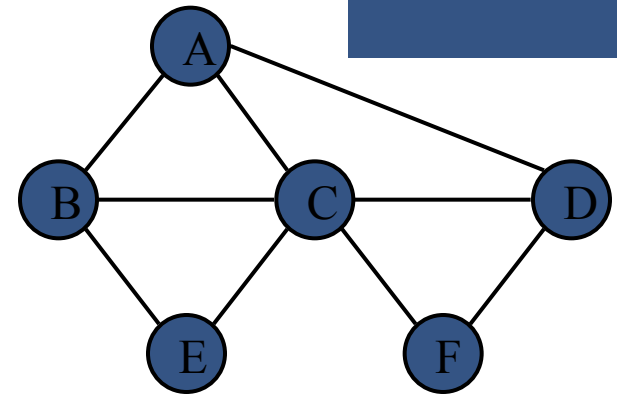
Property 2

The discovery edges labeled by $BFS(G, s)$ form a spanning tree T_s of G_s

Property 3

For each vertex v in L_i

- The path of T_s from s to v has i edges
- Every path from s to v in G_s has at least i edges



Analysis

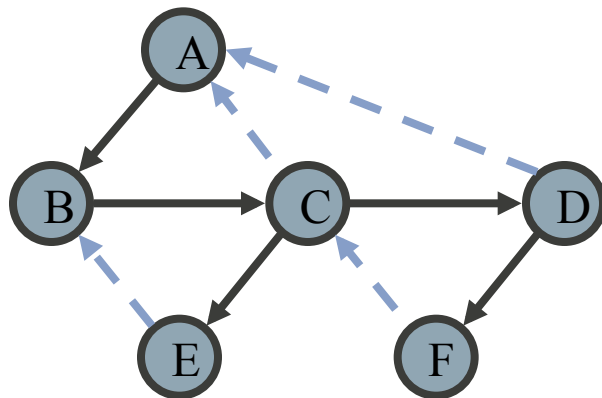
- Setting/getting a vertex/edge label takes $O(1)$ time
- Each vertex is labeled twice
 - once as UNEXPLORED
 - once as VISITED
- Each edge is labeled twice
 - once as UNEXPLORED
 - once as DISCOVERY or CROSS
- Each vertex is inserted once into a sequence L_i
- Method incidentEdges is called once for each vertex
- BFS runs in $O(n + m)$ time provided the graph is represented by the adjacency list structure
 - Recall that $\sum_v \deg(v) = 2m$

Applications

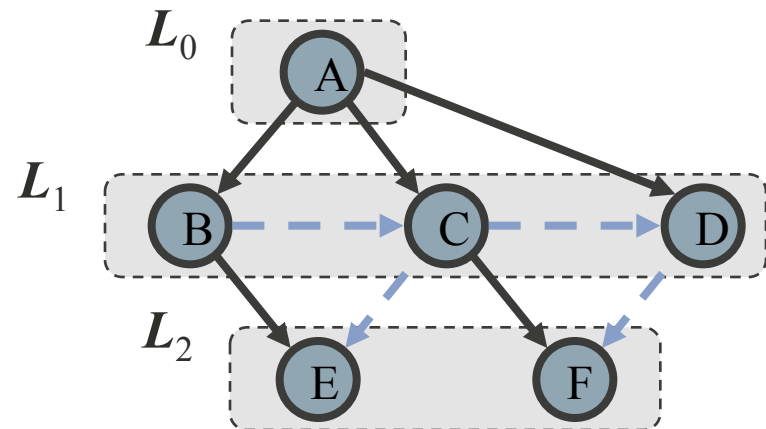
- Using the template method pattern, we can specialize the BFS traversal of a graph G to solve the following problems in $O(n + m)$ time
 - Compute the connected components of G
 - Compute a spanning forest of G
 - Find a simple cycle in G , or report that G is a forest
 - Given two vertices of G , find a path in G between them with the minimum number of edges, or report that no such path exists

DFS vs. BFS

Applications	DFS	BFS
Spanning forest, connected components, paths, cycles	✓	✓
Shortest paths		✓
Biconnected components	✓	



DFS

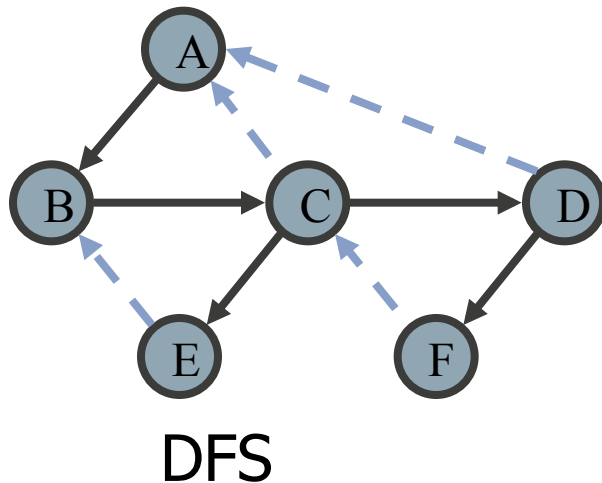


BFS

DFS vs. BFS (cont.)

Back edge (v, w)

- w is an ancestor of v in the tree of discovery edges



Cross edge (v, w)

- w is in the same level as v or in the next level

