アルゴリズムの設計と解析

教授: 黄 潤和 （W4022）
rhuang@hosei.ac.jp

SA: 広野 史明 （A4/A8）
fumiaki.hirono.5k@stu.hosei.ac.jp

Contents (L10 – Review Graph)

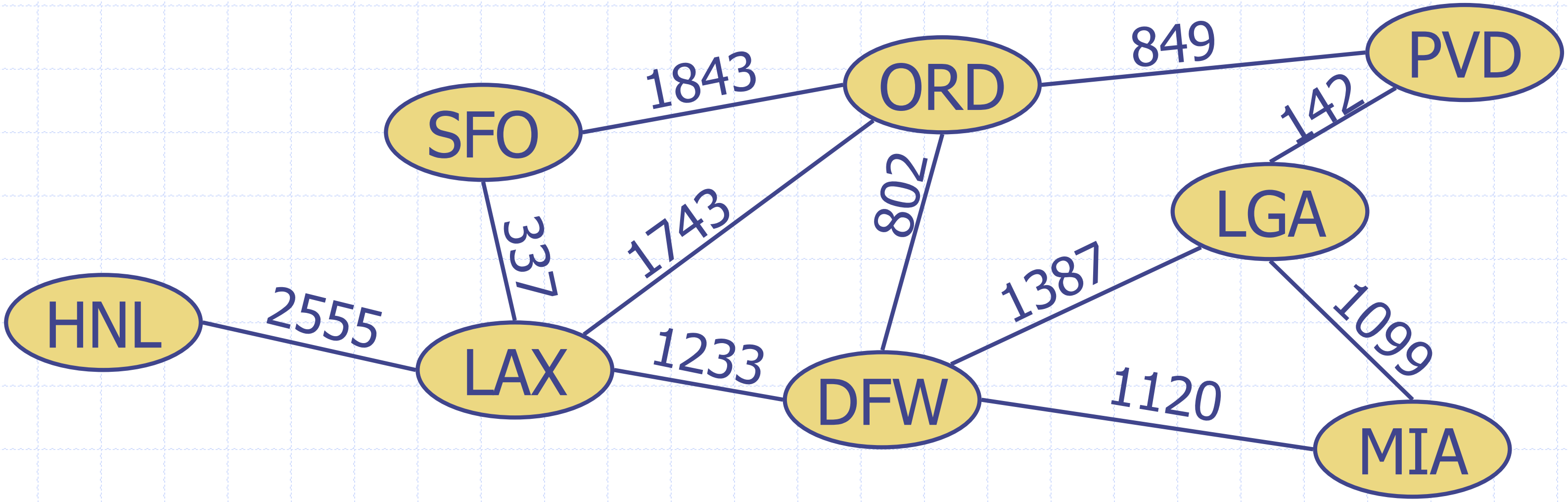
- Basis of Graph
- Depth-First Search

Basis of Graph

- Graphs グラフ
  - Definition 定義
  - Applications アプリケーション
  - Terminology 用語
  - Properties 定義
  - ADT ADT
- Data structures for graphs グラフのためのデータ構造
  - Adjacency list 隣接リスト
  - Adjacency matrix 隣接マトリクス
- Other Concepts
  - Subgraph サブグラフ
  - Connectivity 連結性
  - Spanning trees and forests 全域木、全域森

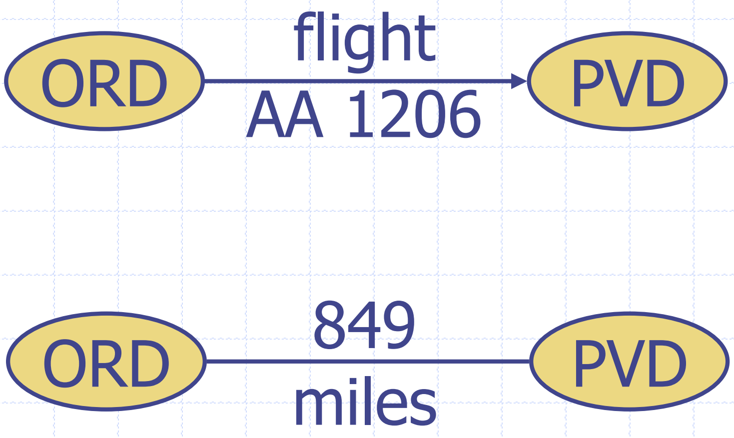
Graph
グラフ

- A graph is a pair (V, E), where
  - V is a set of nodes, called vertices（節、頂点）
  - E is a collection of pairs of vertices, called edges（エッジ、辺、枝）
  - Vertices and edges are positions and store elements
- Example:
  - A vertex represents an airport and stores the three-letter airport code
    節：空港と3文字で表されたその空港名コード
  - An edge represents a flight route between two airports and stores the mileage of the route
    エッジ：2つの空港間の経路とその距離



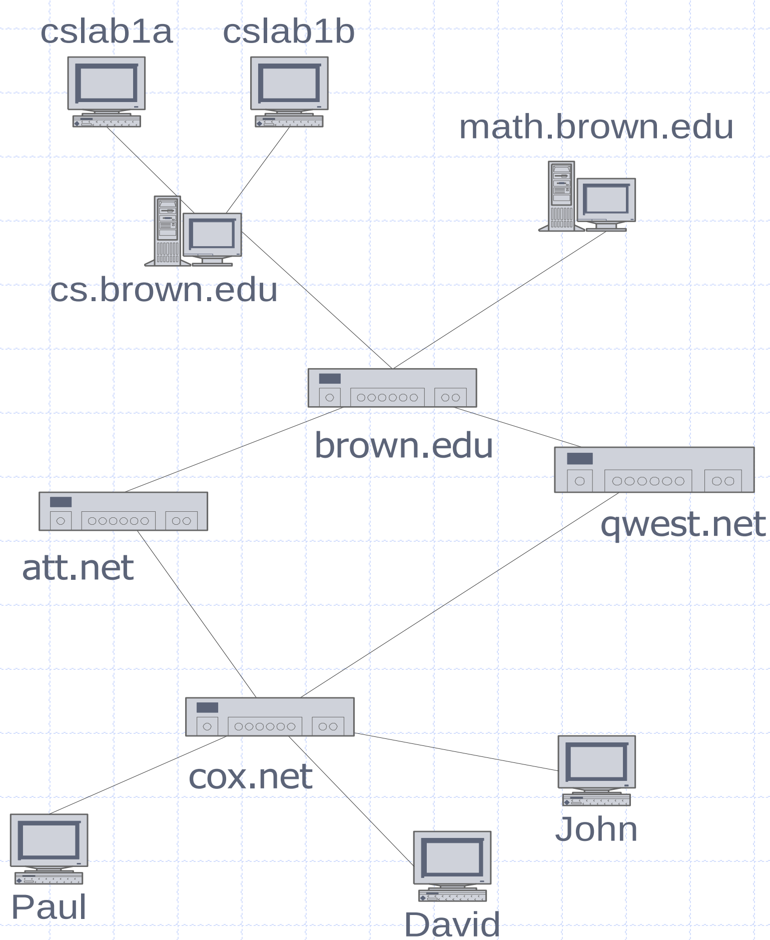
Edge Types
エッジタイプ

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



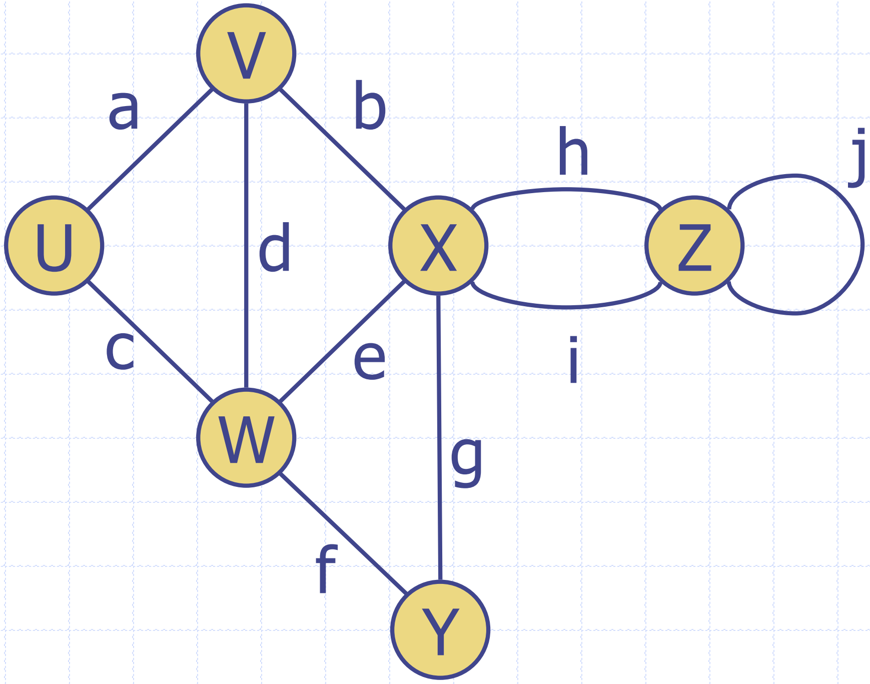
Applications
アプリケーション

- Electronic circuits
  電子回路
  - Printed circuit board
  - Integrated circuit
- Transportation networks
  運送ネットワーク
  - Highway network
  - Flight network
- Computer networks
  コンピュータネットワーク
  - Local area network
  - Internet
  - Web
- Databases データベース
  - Entity-relationship diagram
    ERダイアグラム



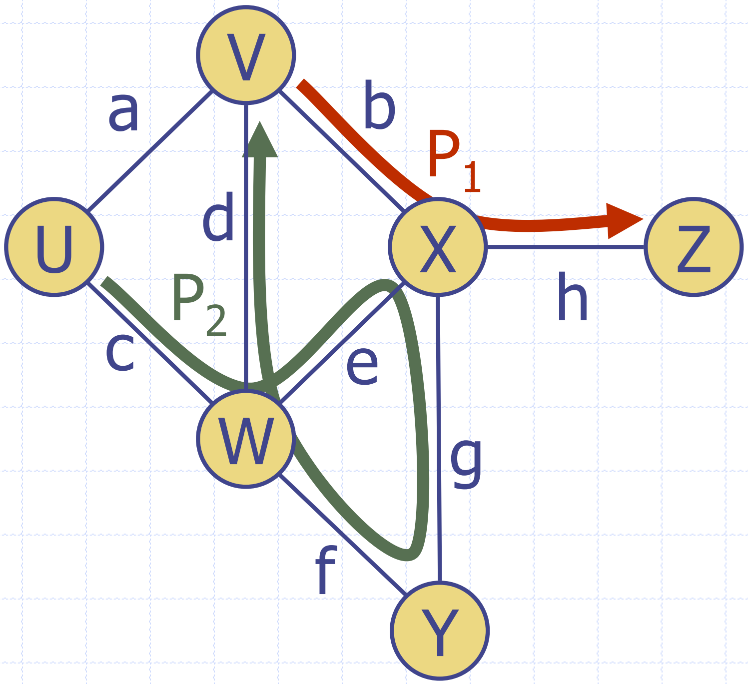
Terminology
用語

- End vertices (or endpoints) of an edge 終点
  - U and V are the endpoints of a
- Edges incident on a vertex 節の接合
  - a, d, and b are incident on V
- Adjacent vertices 隣接
  - U and V are adjacent
- Degree of a vertex 度
  - X has degree 5
- Parallel edges パラレルエッジ
  - h and i are parallel edges
- Self-loop ループ
  - j is a self-loop



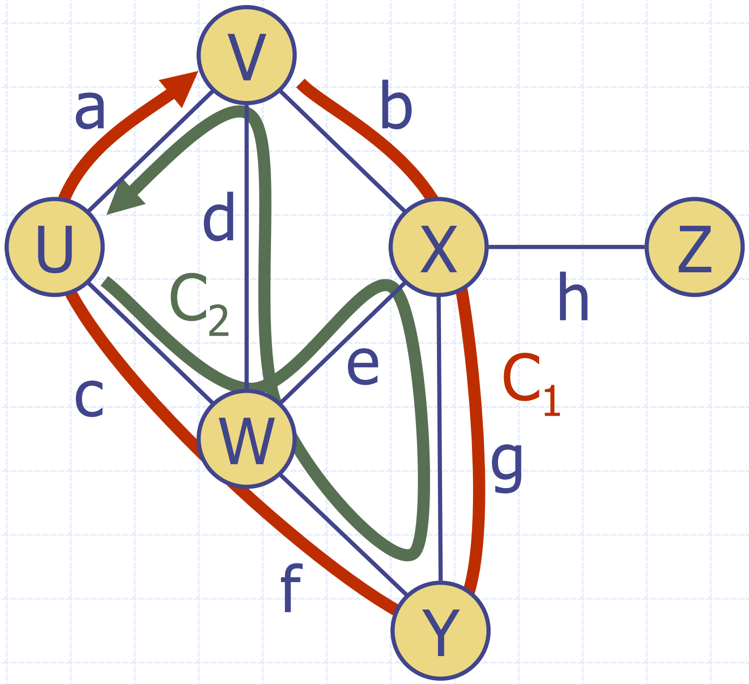
Terminology (cont.)
用語

- 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
    はっきりした、入り組んでないパス
- 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.)
用語

- Cycle サイクル
  - 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,) is a simple cycle
  - C_2=(U,c,W,e,X,g,Y,f,W,d,V,a,) is a cycle that is not simple



Properties
特性

Property 1

$\Sigma_v \deg(v) = 2m$

Proof: each endpoint is counted twice

Property 2

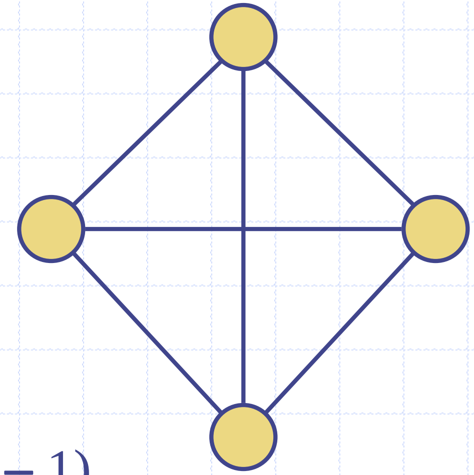
In an undirected graph with no self-loops and no multiple edges

$m \leq n(n-1)/2$

Proof: each vertex has degree at most $(n-1)$

Notation

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



Example

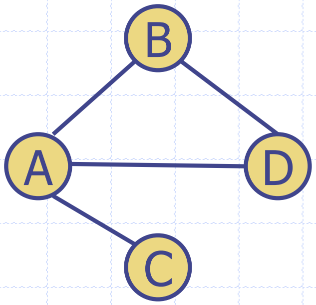
- $n = 4$
- $m = 6$
- $\deg(v) = 3$

Main Methods of the Graph ADT
グラフADTのメインメソッド

- Vertices and edges
  - are positions
  - store elements
- Accessor methods
  - aVertex()
  - incidentEdges(v)
  - endVertices(e)
  - isDirected(e)
  - origin(e)
  - destination(e)
  - opposite(v, e)
  - areAdjacent(v, w)
- Update methods
  - insertVertex(o)
  - insertEdge(v, w, o)
  - insertDirectedEdge(v, w, o)
  - removeVertex(v)
  - removeEdge(e)
- Generic methods
  - numVertices()
  - numEdges()
  - vertices()
  - edges()

Adjacency List
隣接リスト

- An adjacency list is an array of lists. Each individual list shows what vertices a given vertex is adjacent to. 隣接リストは、リストの配列となります。個々のリストは、指定した頂点に隣接する頂点を示している。
- An example: The graph

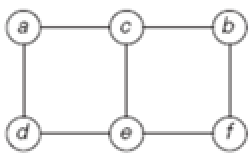


The adjacency list

Vertex	List containing adjacent vertices
A	B → C → D
B	A → D
C	A
D	A → B

Adjacency Matrix
隣接マトリクス

- An adjacency matrix is a two-dimensional array in which the elements indicate whether an edge is present between two vertices. If a graph has *n* vertices, the adjacency matrix is an *n*-by-*n* matrix.
- An example: The graph



	a	b	c	d	e	f
a	0	0	1	1	0	0
b	0	0	1	0	0	1
c	1	1	0	0	1	0
d	1	0	0	0	1	0
e	0	0	1	1	0	1
f	0	1	0	0	1	0

(a)

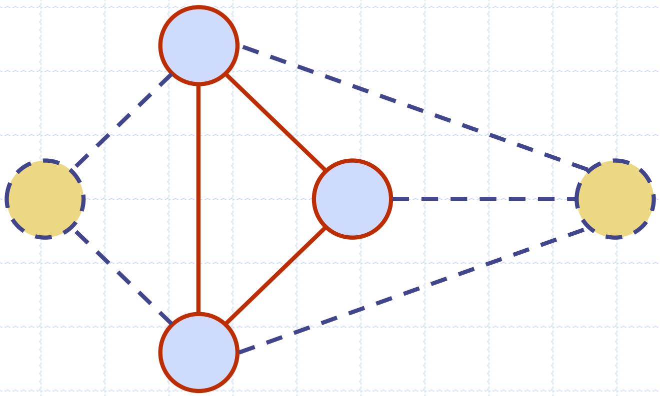
| a | → c → d
| b | → c → f
| c | → a → b → e
| d | → a → e
| e | → c → d → f
| f | → b → e

(b)

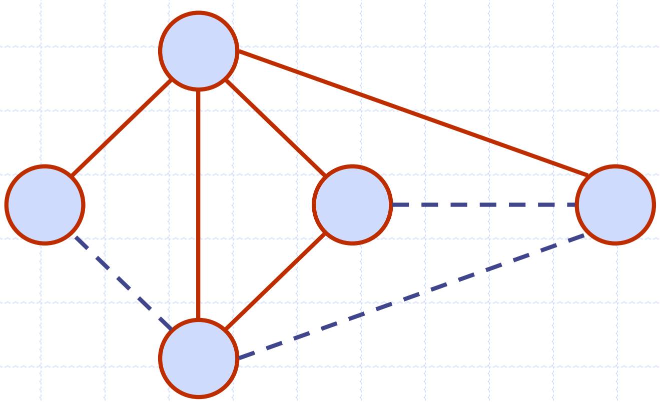
FIGURE 1.7 (a) Adjacency matrix and (b) adjacency lists of the graph in Figure 1.6a.

Subgraphs
サブグラフ

- A subgraph S of a graph G is a graph such that
  - The edges of S are a subset of the edges of G
  - The vertices of S are a subset of the vertices of G
- A spanning subgraph of G is a subgraph that contains all the vertices of G
  Gの全域部分木:全ての節を含む



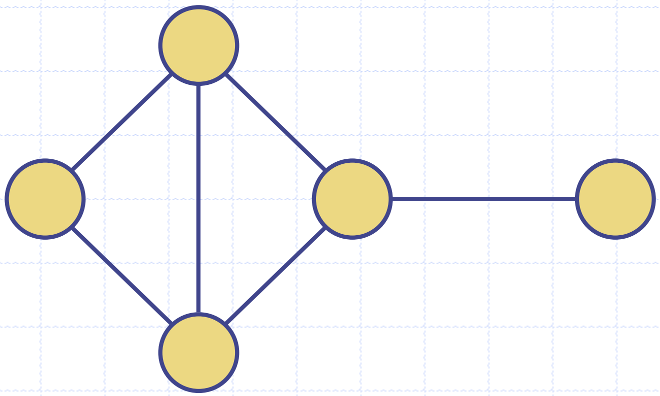
Subgraph



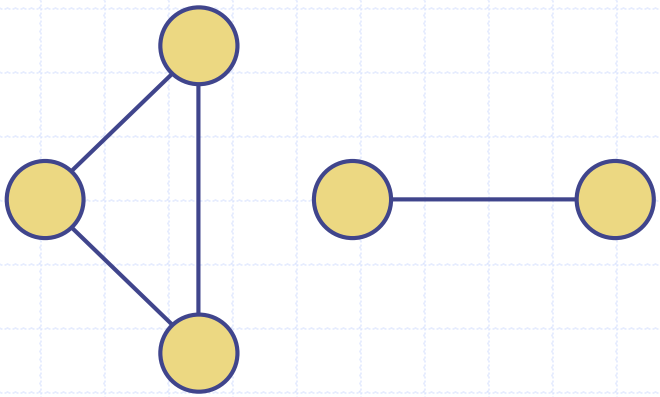
Spanning subgraph

Connectivity
連結性

- A graph is connected if there is a path between every pair of vertices
  連結グラフ: 全ての節がお互いに接続されている。
- A connected component of a graph G is a maximal connected subgraph of G
  連結部位はグラフGの最大のサブグラフである。



Connected graph

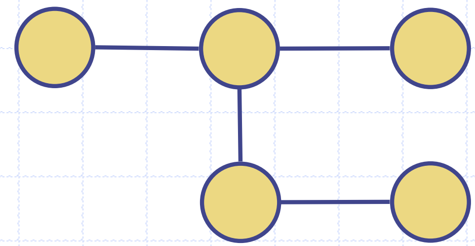


Non connected graph with two connected components

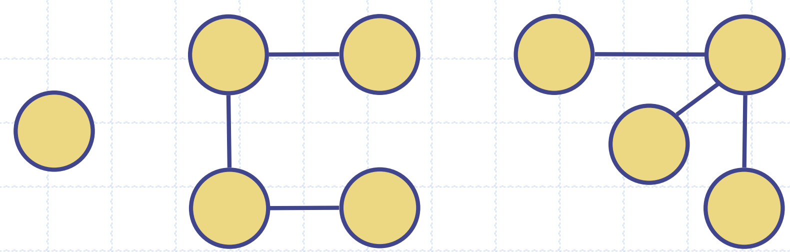
Trees and Forests
木と森

- A (free) tree is an undirected graph T such that
  - T is connected　連結している
  - T has no cycles　サイクルがない

  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
  森の連結部位はすべて木



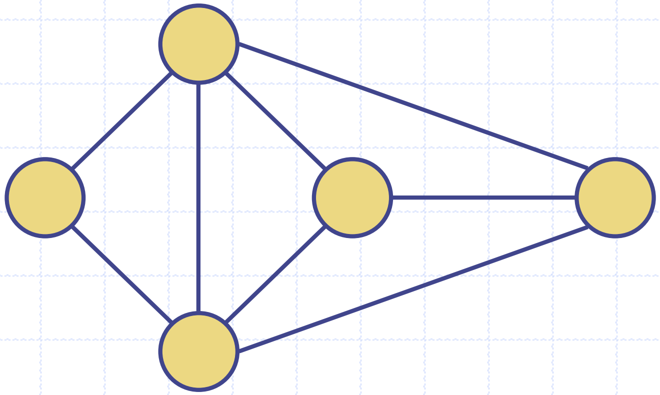
Tree



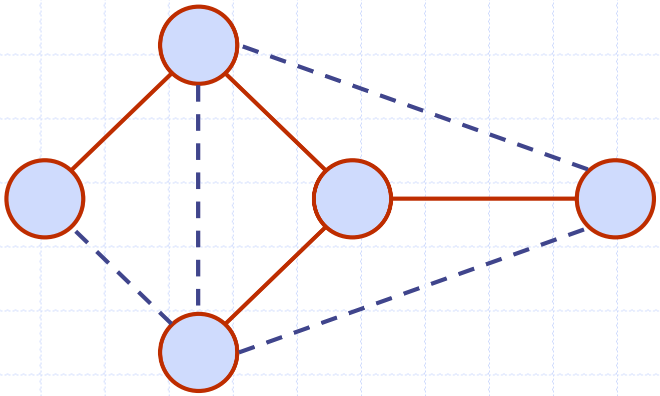
Forest

Spanning Trees and Forests 全域木と全域森

- 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
グラフが木でない限り全域木は1つではない。
- Spanning trees have applications to the design of communication networks
コミュニケーションネットワークへの利用
- A spanning forest of a graph is a spanning subgraph that is a forest

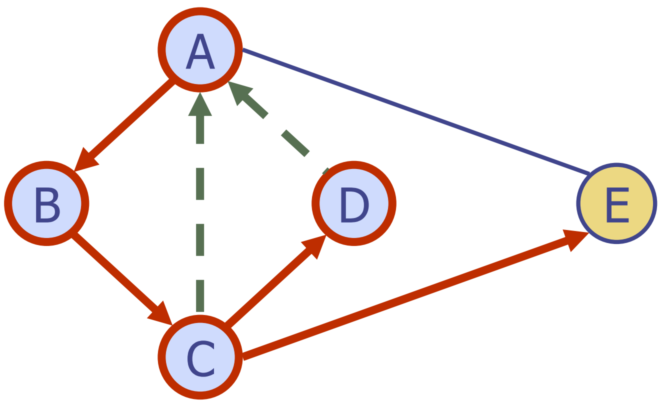


Graph



Spanning tree

Depth-First Search
深さ優先探索

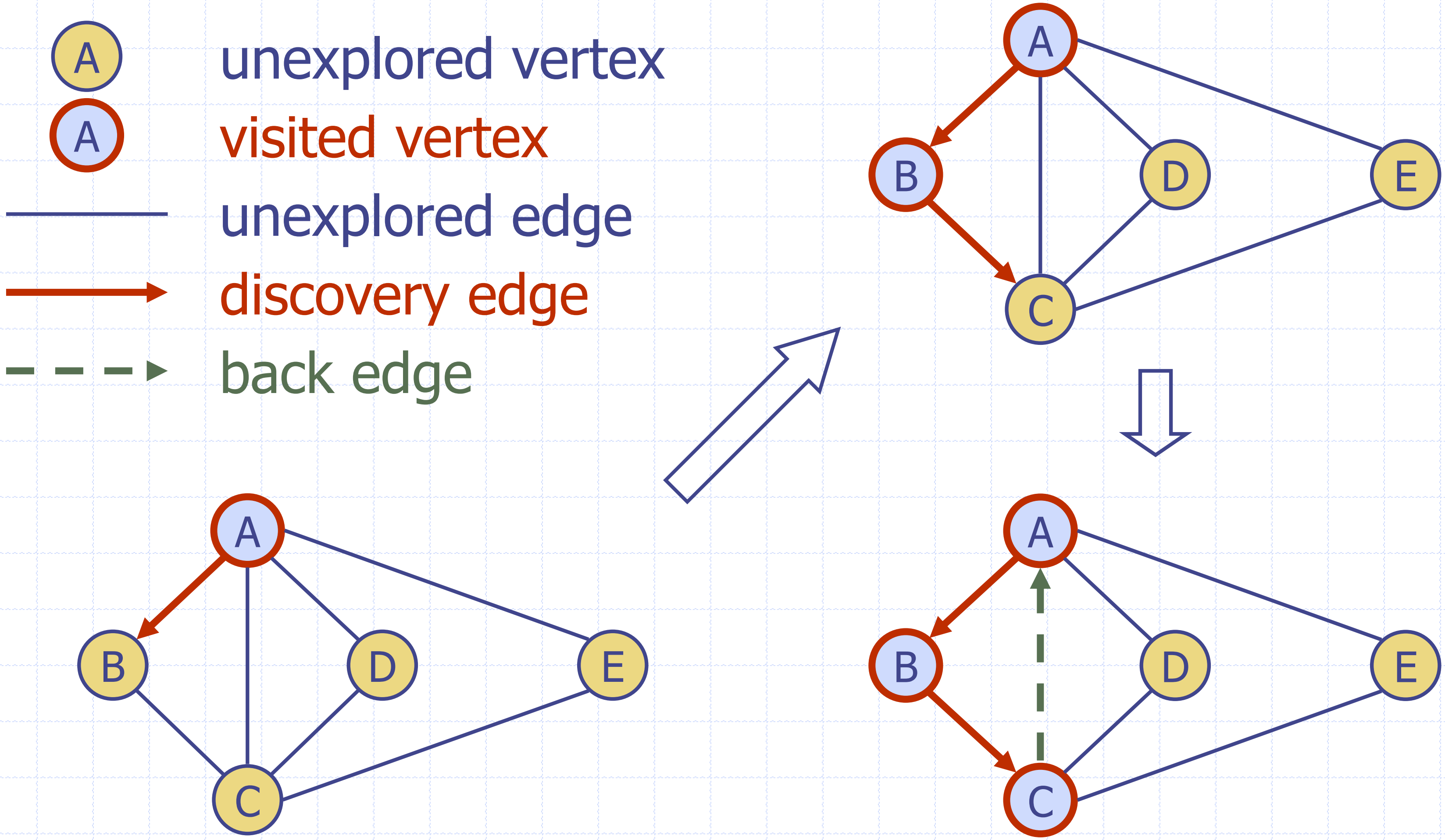


Outline

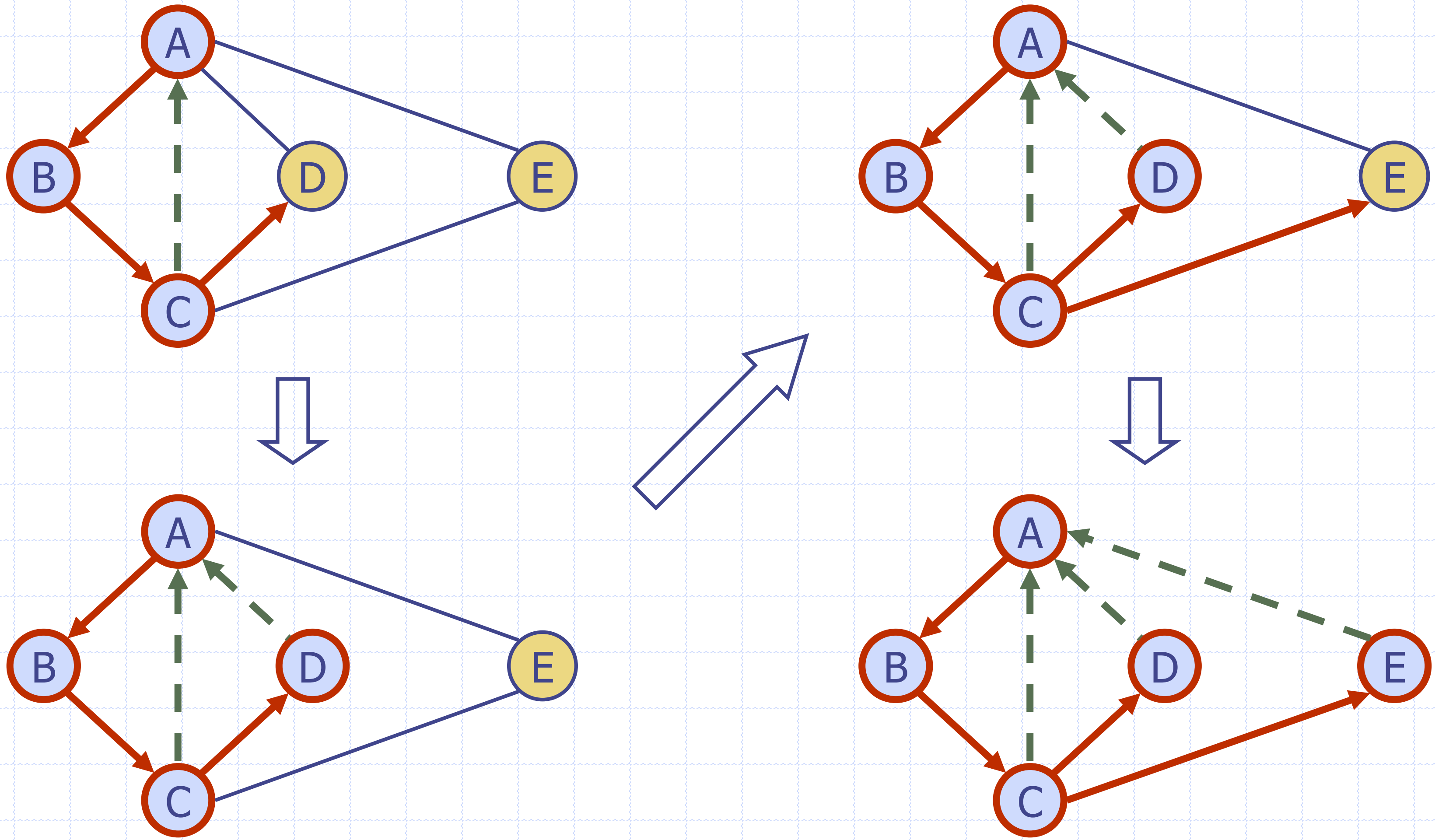
- Depth-first search 深さ優先探索
  - Algorithm アルゴリズム
  - Example 例
  - Properties 特性
  - Analysis 分析
- Applications of DFS DFSのアプリケーション
  - Path finding 経路調査結果
  - Cycle finding サイクル調査結果

Example

例　unexplored : 未訪問　visited : 訪問済

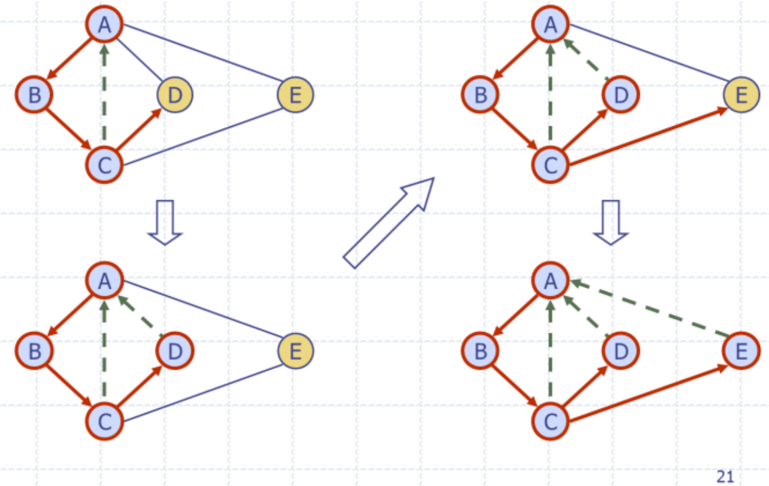


Example (cont.)
例



DFS Algorithm
DFSアルゴリズム

- 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 ∈ G.vertices()
    setLabel(u, UNEXPLORED)
  for all e ∈ G.edges()
    setLabel(e, UNEXPLORED)
  for all v ∈ 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 ∈ G.incidentEdges(v)
    if getLabel(e) = UNEXPLORED
      w ← opposite(v,e)
      if getLabel(w) = UNEXPLORED
        setLabel(e, DISCOVERY)
        DFS(G, w)
      else
        setLabel(e, BACK)
```

Depth-First Search
深さ優先探索

- Depth-first search (DFS) is a general technique for traversing a graph
  グラフ探索の一般的な手法の1つ
- A DFS traversal of a graph G
  - Visits all the vertices and edges of G
    全ての節と枝を訪れる
  - Determines whether G is connected
    Gが連結しているかの判断
  - Computes the connected components of G
    Gの接続部位の計算
  - Computes a spanning forest of G
    全域森の計算
- DFS on a graph with n vertices and m edges takes $O(n + m)$ time
  n個の節とm個の枝の場合の時間: $O(n + m)$
- DFS can be further extended to solve other graph problems
  - Find and report a path between two given vertices
    与えられた2点間のパスの探索と表示
  - Find a cycle in the graph
    グラフ内のサイクルの発見

Properties of DFS
DFSの特性

Property 1

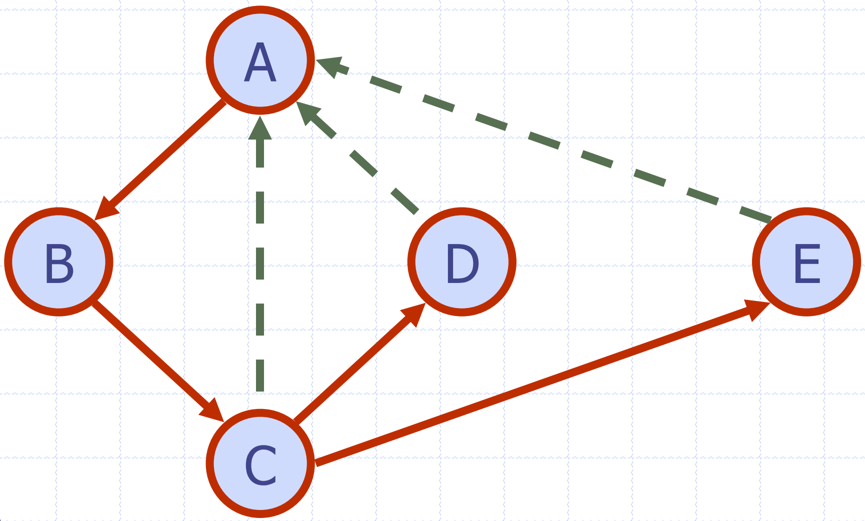
DFS(*G*, *v*) visits all the vertices and edges in the connected component of *v*

全ての節と枝を訪れる。

Property 2

The discovery edges labeled by *DFS*(*G*, *v*) form a spanning tree of the connected component of *v*

訪問済みの枝はラベルを貼られる。



Analysis of DFS
DFSの分析

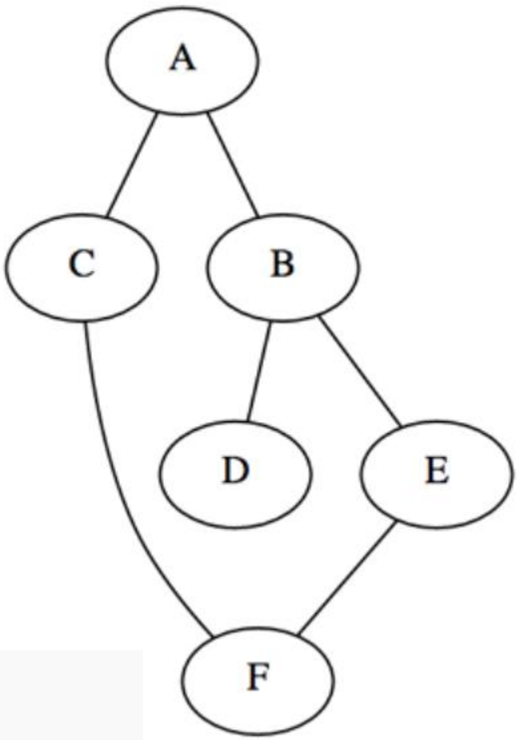
- Setting/getting a vertex/edge label takes $O(1)$ time
  節や枝のラベルの設定や取得: $O(1)$
- Each vertex is labeled twice
  - once as UNEXPLORED（未訪問）
  - once as VISITED（訪問済）
- Each edge is labeled twice
  - once as UNEXPLORED
  - once as DISCOVERY or BACK（発見されたor戻る）
- Method incidentEdges is called once for each vertex
- DFS runs in $O(n + m)$ time provided the graph is represented by the adjacency list structure
  実行時間: $O(n + m)$
  - Recall that $\Sigma_v \deg(v) = 2m$

Work in class

Find the references (Python or Java implementation of DSF)

In Python

```
graph = {'A': set(['B', 'C']),
         'B': set(['A', 'D', 'E']),
         'C': set(['A', 'F']),
         'D': set(['B']),
         'E': set(['B', 'F']),
         'F': set(['C', 'E'])}
```



Below is a listing of the actions performed upon each visit to a node.

- Mark the current vertex as being visited.
- Explore each adjacent vertex that is not included in the visited set.

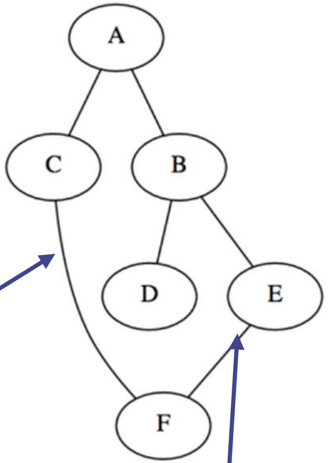
using the stack data-structure

```
def dfs(graph, start):
    visited, stack = set(), [start]
    while stack:
        vertex = stack.pop()
        if vertex not in visited:
            visited.add(vertex)
            stack.extend(graph[vertex] - visited)
    return visited

dfs(graph, 'A') # {'E', 'D', 'F', 'A', 'C', 'B'}
```

output

Returning all possible paths between a start and goal vertex.

```
def dfs_paths(graph, start, goal):
    stack = [(start, [start])]
    while stack:
        (vertex, path) = stack.pop()
        for next in graph[vertex] - set(path):
            if next == goal:
                yield path + [next]
            else:
                stack.append((next, path + [next]))
```

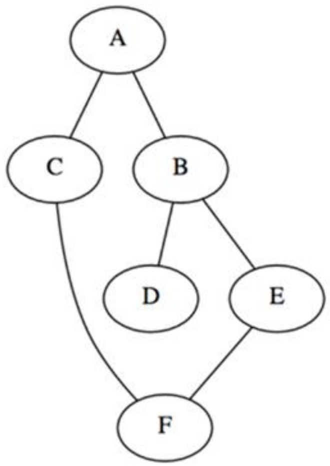



```
list(dfs_paths(graph, 'A', 'F')) # [['A', 'C', 'F'], ['A', 'B', 'E', 'F']]
```

Recursive approach

```
def dfs(graph, start, visited=None):
    if visited is None:
        visited = set()
    visited.add(start)
    for next in graph[start] - visited:
        dfs(graph, next, visited)
    return visited

dfs(graph, 'C')
```

```
graph = {'A': set(['B', 'C']),
         'B': set(['A', 'D', 'E']),
         'C': set(['A', 'F']),
         'D': set(['B']),
         'E': set(['B', 'F']),
         'F': set(['C', 'E'])}

def dfs_paths(graph, start, goal, path=None):
    if path is None:
        path = [start]
    if start == goal:
        yield path
    for next in graph[start] - set(path):
        yield from dfs_paths(graph, next, goal, path + [next])

list(dfs_paths(graph, 'C', 'F')) # [['C', 'F'], ['C', 'A', 'B', 'E', 'F']]
```



In Java

http://www.algolist.net/Algorithms/Graph/Undirected/Depth-first_search

Initially all vertices are white (unvisited). DFS starts in arbitrary vertex and runs as follows:

1. Mark vertex **u** as gray (visited).
2. For each edge **(u, v)**, where **u** is white, run depth-first search for **u** recursively.
3. Mark vertex **u** as black and backtrack to the parent.

Java

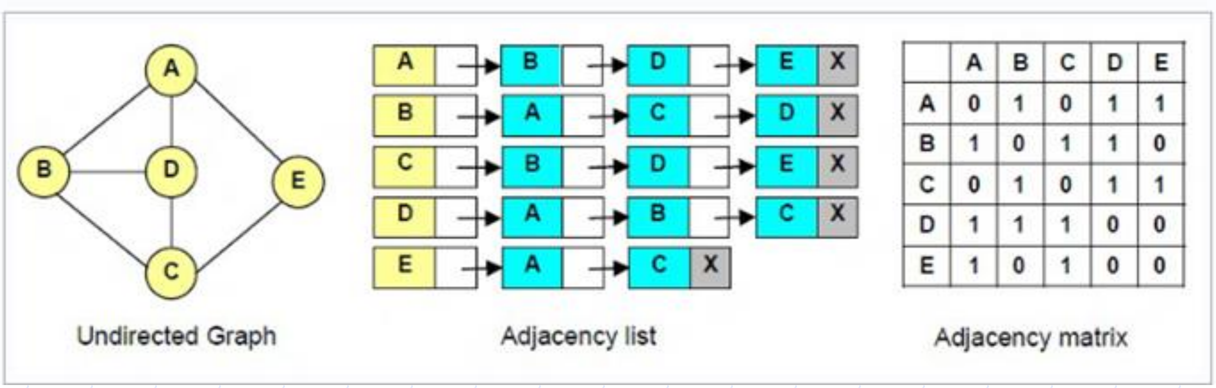
```
public class Graph {
      ...

      enum VertexState {
            White, Gray, Black
      }

      public void DFS()
      {
            VertexState state[] = new VertexState[vertexCount];
            for (int i = 0; i < vertexCount; i++)
                  state[i] = VertexState.White;
            runDFS(0, state);
      }

      public void runDFS(int u, VertexState[] state)
      {
            state[u] = VertexState.Gray;
            for (int v = 0; v < vertexCount; v++)
                  if (isEdge(u, v) && state[v] == VertexState.White)
                        runDFS(v, state);
            state[u] = VertexState.Black;
      }
}
```

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
      $setLabel(e, BACK)$

http://sourcecode4all.wordpress.com/tag/depth-first-search/
http://sourcecode4all.com/depth-first-search/

Use adjacent list to implement DFS

```
public void dfs(int head) // recursive depth-first search
{
    Node w;
    int v;
    mark[head] = 1; // 1 : if node v is already visited, 0 : if not
    System.out.print(head + " ");
    w = adjList[head];          // adjList is adjacent list
    while (w != null) {
        v = w.label;
        if (mark[v] == 0)
            dfs(v);
        w = w.next;
    }
}
```



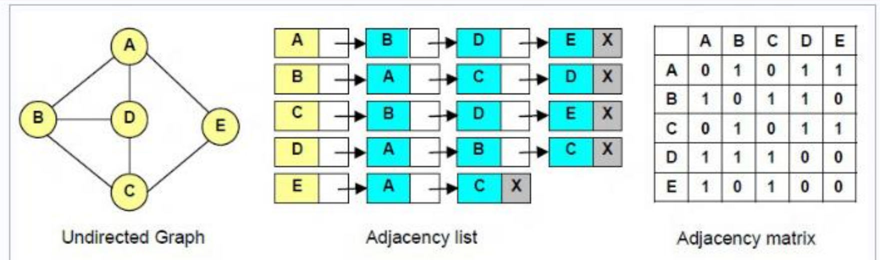
http://sourcecode4all.wordpress.com/tag/depth-first-search/

```java
public void dfs(int head) // recursive depth-first search
{
    Node w;
    int v;
    mark[head] = 1; // 1 : if node v is already visited, 0 : if not
    System.out.print(head + " ");
    w = adjList[head];
    while (w != null) {
        v = w.label;
        if (mark[v] == 0)
            dfs(v);
        w = w.next;
    }
```

Depth-First Search

```java
class Node
{ int label; // vertex label
Node next; // next node in list
Node( int b ) // constructor
{ label = b; }
}
class Graph
{ int size;
Node adjList[];
int mark[];
Graph(int n) // constructor
{ size = n;

adjList = new Node[size];
mark = new int[size]; // elements of mark are initialized to 0
}
```

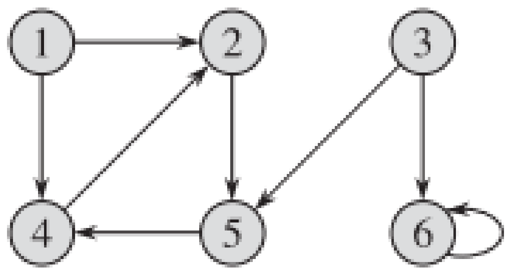
```java
public void createAdjList(int a[][]) // create adjacent lists
{
Node p; int i, k;
for( i = 0; i < size; i++ )
{ p = adjList[i] = new Node(i); //create first node of ith adj. list
for( k = 0; k < size; k++ )
{ if( a[i][k] == 1 )
{ p.next = new Node(k); // create next node of ith adj. list
p = p.next;
}}}}
```



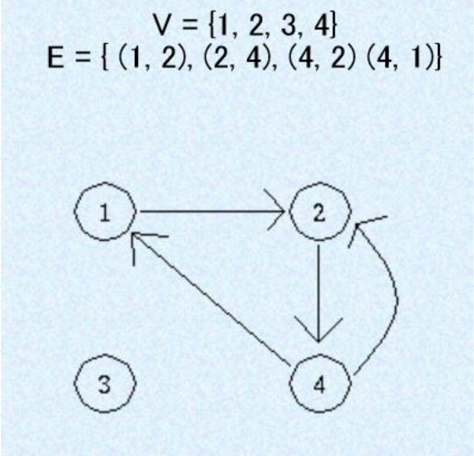
```
public void dfs(int head) // recursive depth-first search
{
    Node w;
    int v;
    mark[head] = 1; // 1 : if node v is already visited, 0 : if not.
    System.out.print(head + " ");
    w = adjList[head];          // adjList is adjacent list
    while (w != null) {
        v = w.label;            // label is the label of a vertex
        if (mark[v] == 0)
            dfs(v);
        w = w.next;
    }
}
```

http://lab.tomires.eu/metro/

Work in class



A directed graph G with 6 vertices and 8 edges,
Please write (1) an adjacency-list representation of G.
(2) The adjacency-matrix representation of G.

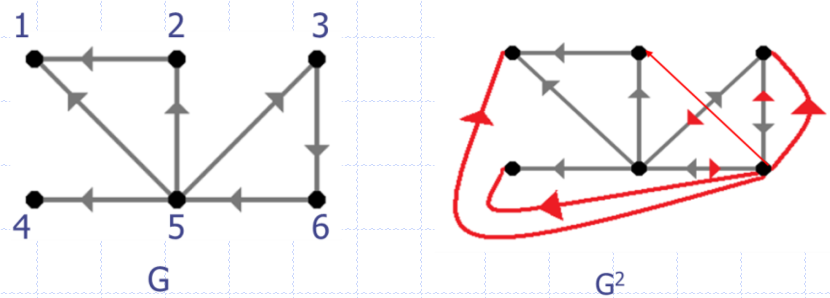


Define a graph G=(V, E),

for example, V = {1, 2, 3, 4}
E = { (1, 2), (2, 4), (4, 2) (4, 1)}

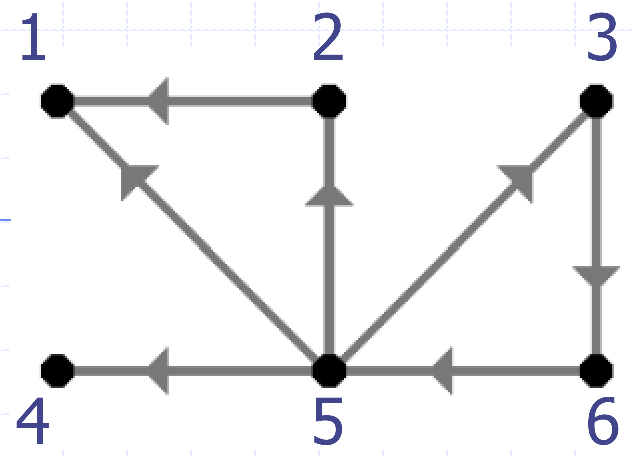
1. Transpose
If graph G = (V, E) is a directed graph, its transpose, GT = (V, ET) is the same as graph G with all arrows reversed.

2. Square
The square of a directed graph G = (V, E) is the graph $G^2 = (V, E^2)$ such that $(a, b) \in E^2$ if and only if for some vertex $c \in V$, both $(u, c) \in E$ and $(c,b) \in E$. That is, G^2 contains an edge between vertex *a* and vertex *b* whenever G contains a path with exactly two edges between vertex *a* and vertex *b*.



An example

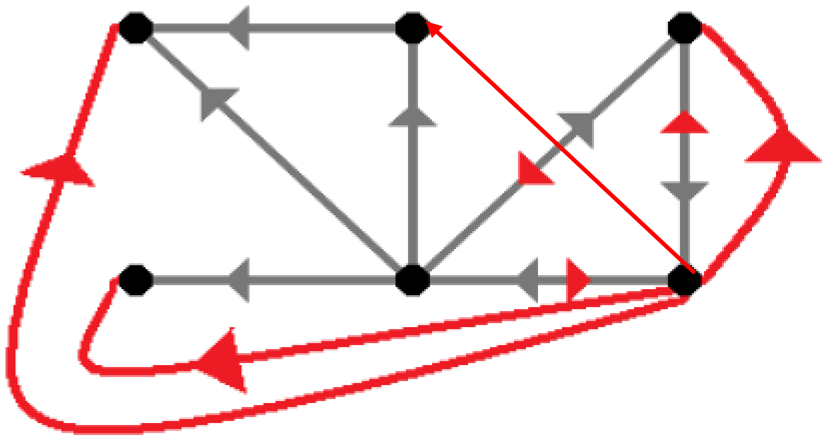
Ex. 10-1 and 10-2 (Work in class)
What the algorithms in pseudo codes for
1. Graph Transpose
2. Graph Square



G

If we label the vertices 1 to 6 (top three are 1, 2 and 3, bottom three from left to right are 4, 5 and 6), we get the following adjacency matrix:

	1	2	3	4	5	6
1	0	0	0	0	0	0
2	1	0	0	0	0	0
3	0	0	0	0	0	1
4	0	0	0	0	0	0
5	1	1	1	1	0	0
6	0	0	0	0	1	0



	1	2	3	4	5	6
1	0	0	0	0	0	0
2	1	0	0	0	0	0
3	0	0	0	0	1	1
4	0	0	0	0	0	0
5	1	1	1	1	0	1
6	1	1	1	1	1	0

The arc (5,1) is not doubled up because it already exists.

G^2