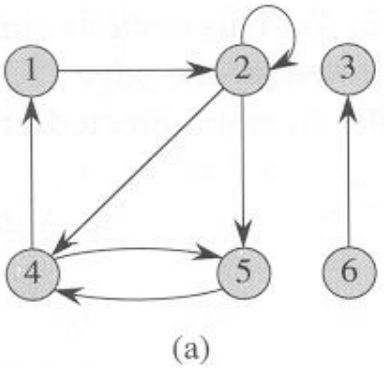
Elementary Graph Algorithm

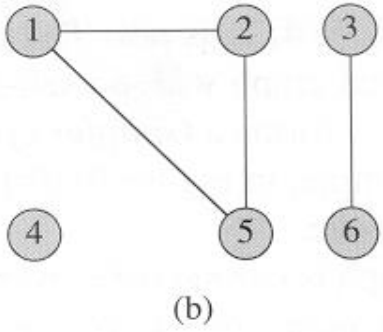
Graph

- G = (V, E)
  - V : set of vertices
  - E : set of edges
    - Directed graph
    - Undirected graph



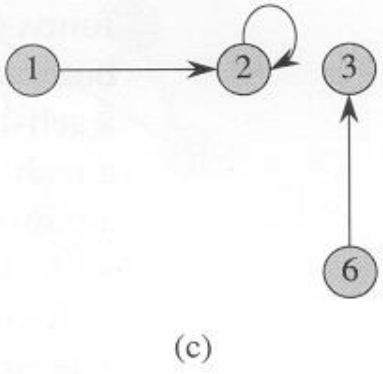
(a)

V={1,2,3,4,5,6}
E = {(1,2), (2,2), (2,4), (2,5),
(4,1), **(4,5)**, **(5,4)**, (6,3)}



(b)

V={1,2,3,4,5,6}
E = {(1,2), (1,5), (2,4), (3,6)}



(c)

Sub-graph of (a)

* Self-loop

Graph

- Applications

Graph	Vertices	Edges
통신	전화, 컴퓨터	광섬유 케이블
전기전자 회로	칩, 콘덴서, 저항	선
기계	이음새	스프링, 빔, 막대
급수시스템	저수지, 정화시설	배관
교통	교차로, 공항	고속도로, 항로
일정표	단위작업	선결 조건
소프트웨어 시스템	함수	함수호출
인터넷	웹 페이지	하이퍼 링크
게임	말의 위치	허용된 움직임
분자구조	분자	화학적 연결
사회적 관계	사람	친구관계
경제	주식, 현금	거래

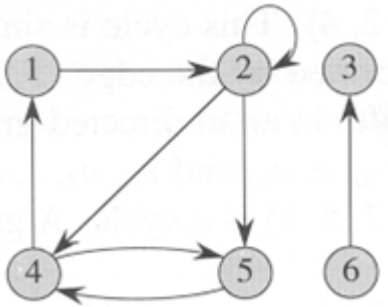
Graph

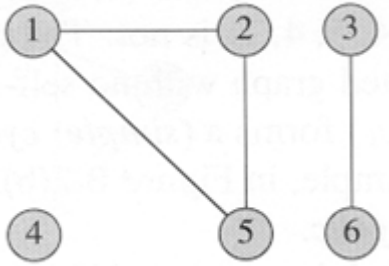
- Incident (입사, 투사)
  - Edge of digraph: (u, v)
    - incident from u
    - incident to v

  (ex) edges *incident from* vertex 2:

  (ex) edges *incident to* vertex 2:
  - Edge of undirected graph: (u, v)
    - incident on u and v

  (ex) edges *incident on* vertex 2:





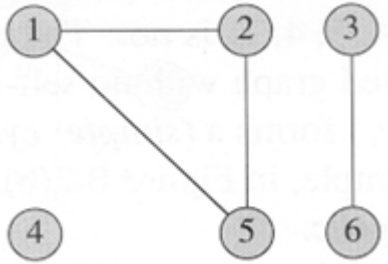
- Adjacent
  - Edge of graph: (u, v)
    - Vertex v is *adjacent to* vertex u

  (ex) edge (1, 2):

  ❖ Undirected graph: symmetric relation

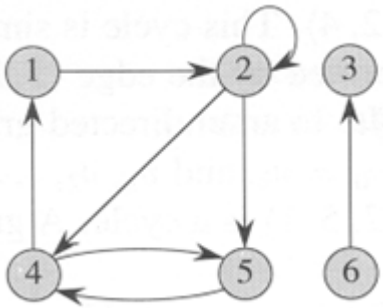
Graph

- Degree of a vertex
  - Undirected graph
    : number of edges *incident on* it
    (ex)



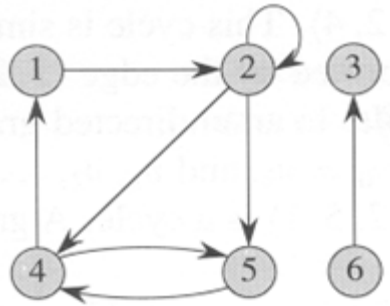
  - Directed graph
    - Out-degree
      : number of edges *incident from* it
    - In-degree
      : number of edges *incident to* it
    - Degree = out-degree + in-degree

    (ex)



Graph

- Path
  - Sequence of vertices: $\langle v_o, v_1, \ldots, v_k \rangle$, $(v_{i-1}, v_i) \in E$
  - Length of a path
    : number of edges in the path



- Cycle
  - Path $\langle v_o, v_1, \ldots, v_k \rangle$ if $v_o = v_k$
- Reachable

  Vertex u′ is reachable from vertex u

  ↔ there is a path from u to u′

  ↔ $u \rightsquigarrow u'$
- Connected

  A graph is connected

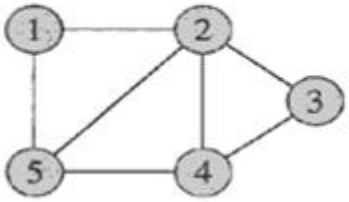
  ↔ every vertex is reachable from every other vertex

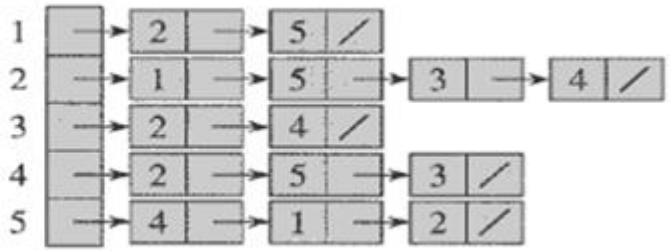
Representation of Graphs

- Adjacency List (인접 리스트)
  - Graph 를 Linked list 를 사용하여 표현하는 방법
  - Adj[*u*], *u*∈V

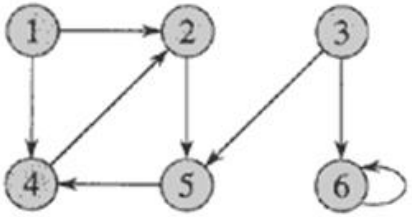
    For each *u*∈V, Adj[*u*] contains all vertices *v* such that there is an edge (*u*, *v*) ∈ E

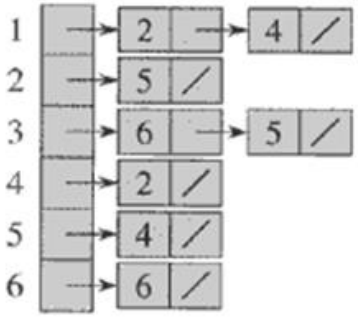
(ex) undirected graph





(ex) digraph





Representation of Graphs

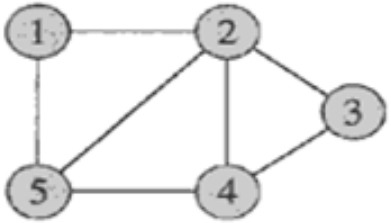
- Adjacency List (인접 리스트)
  - 모든 adjacency list 길이의 합
    - Undirected graph: |E|
    - Directed graph: 2 |E|
  - Memory requirements
    - Θ (|V| + |E|)
    - Sparse graph (|E| << $|V|^2$) 의 표현에 적합
  - Edge (u, v) 존재 여부 확인이 복잡하다

  * large graph 에 주로 적용

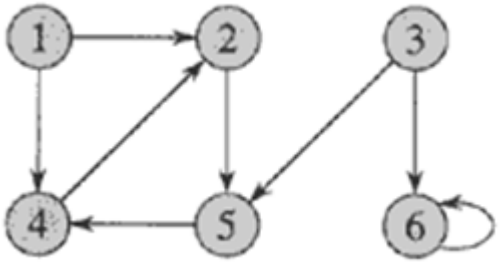
Representation of Graphs

- Adjacency Matrix (인접 행렬)
  - 2 dim. Array 를 사용하여 그래프 표현
  - A = (a_{ij}) (i=1,...|V|, j=1,...|V|)

$$a_{ij} = \begin{cases} 1 & \text{if } (i,j) \in E \\ 0 & \text{otherwise} \end{cases}$$



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



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

Representation of Graphs

- Adjacency Matrix
  - Undirected graph: $A = A^T$
  - Memory requirements
    - $\Theta(|V|^2)$
    - Dense graph ($|E| \fallingdotseq |V|^2$) 의 표현에 적합
  - Edge (u, v) 존재 여부 확인이 간단하다

  * Small graph 에 주로 적용

Breadth-First Search

- 목적
  - 주어진 graph G=(V,E) 와 source vertex s 에 대하여, s 에서 reachable 한 모든 vertex 탐색
  - s 로 부터 모든 reachable vertex 까지의 distance (smallest number of edges) 계산
- Data

  Adj[u]: graph 의 adjacency list

  color[u]: 각 vertex 의 status

  white - not discovered

  gray – discovered but not confirmed

  black – discovered and confirmed

  d[u]: s 부터 u 까지의 distance

  p[u]: u 의 predecessor (or parent)

  (u, v) 가 edge 인 경우, u 는 v 의 predecessor

  Q: queue (first-in first-out)

Breadth-First Search

- Idea
  - Send a wave out from s.
    - First hits all vertices 1 edge from s.
    - From there, hits all vertices 2 edges from s.
    - Etc.
  - Use FIFO queue Q to maintain the wavefront
    - $v \in Q$ if and only if wave has hit v but has not come out of v yet.

Breadth-First Search

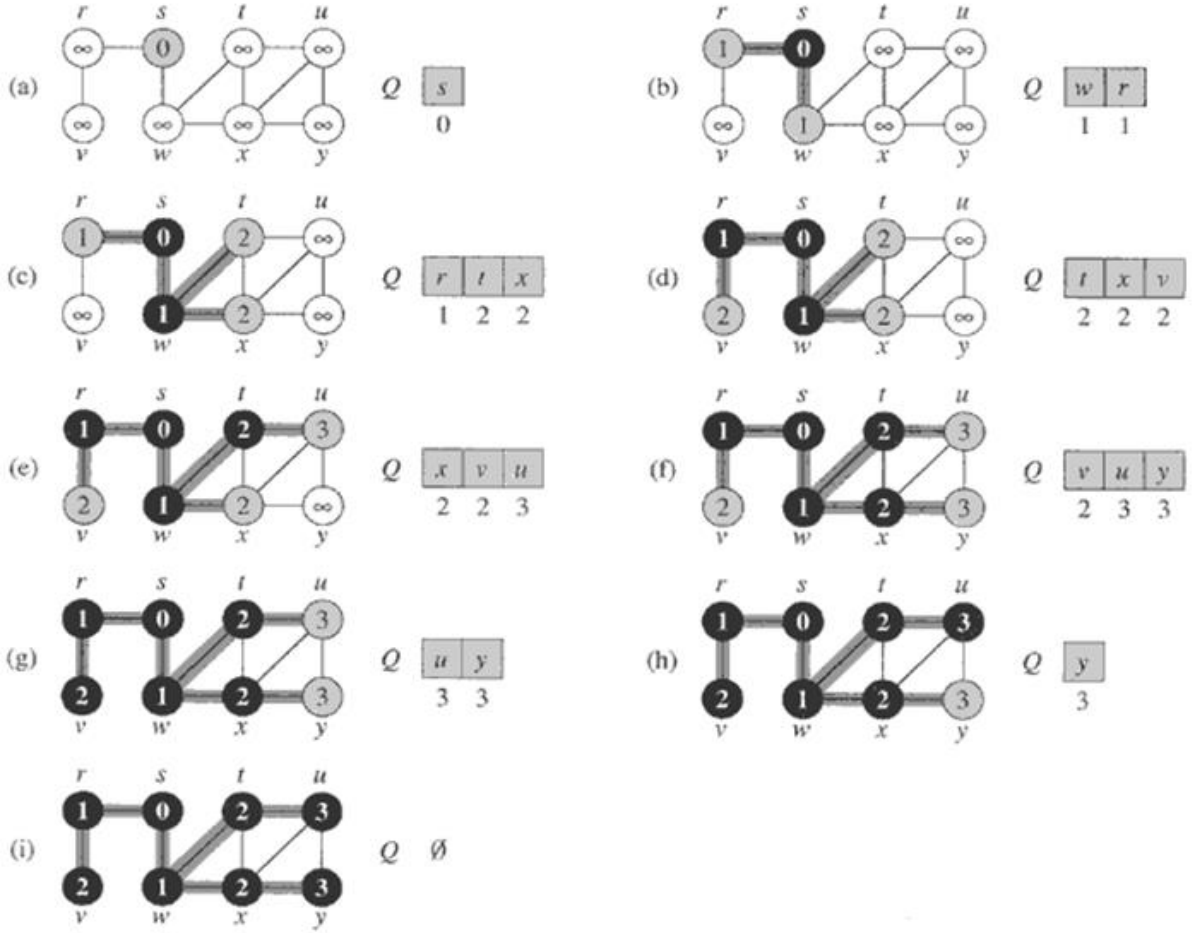
- BFS(G, s)

```
BFS(G, s)
 1  for each vertex u ∈ V[G] − {s}
 2       do color[u] ← WHITE
 3          d[u] ← ∞
 4          π[u] ← NIL
 5  color[s] ← GRAY
 6  d[s] ← 0
 7  π[s] ← NIL
 8  Q ← ∅
 9  ENQUEUE(Q, s)
10  while Q ≠ ∅
11       do u ← DEQUEUE(Q)
12          for each v ∈ Adj[u]
13               do if color[v] = WHITE
14                     then color[v] ← GRAY
15                          d[v] ← d[u] + 1
16                          π[v] ← u
17                          ENQUEUE(Q, v)
18          color[u] ← BLACK
```

* running time: O (V+E)
- 초기화: vertex 수에 비례 O(V)
- loop: edge 수에 비례 O(E)

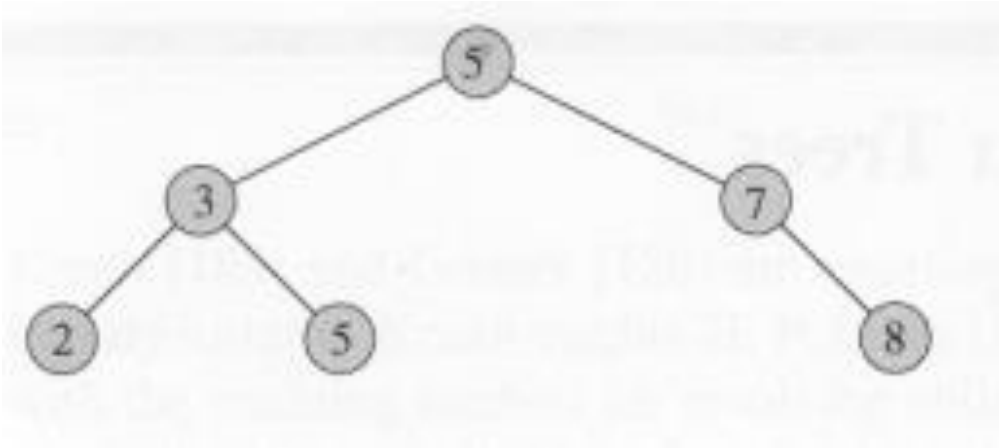
Breadth-First Search

- Example



Breadth-First Search

(Q)



Breadth-First Search

- PRINT-PATH(G,s,v)
  - s 부터 v 까지의 최단 path (breadth-first path) print
  - p[v] 를 trace 하며 인쇄

```
PRINT-PATH(G, s, v)
1  if v = s
2     then print s
3     else if π[v] = NIL
4             then print "no path from" s "to" v "exists"
5             else PRINT-PATH(G, s, π[v])
6                  print v
```

Depth-First Search

- 목적
  - 주어진 graph G=(V,E) 의 모든 vertex 발견 (discover)
  - 각 vertex 에 대한 time stamps (discovery time, finish time) 기록
- Data

  Adj[u]: graph 의 adjacency list

  color[u]: 각 vertex 의 status

  white - not discovered

  gray – discovered but not finished

  black – discovered and finished

  d[u]: discovery time

  f[u]: finish time (d[u] < f[u])

  p[u]: u 의 predecessor (or parent)

  Stack

Depth-First Search

- Idea
  - Tree 의 pre-order traverse 와 유사
  - discover: v1 → v2=adj[v1] → v3=adj[v2] →
  - finish: … → v3 → v2 → v1 방식
- DFS(G)

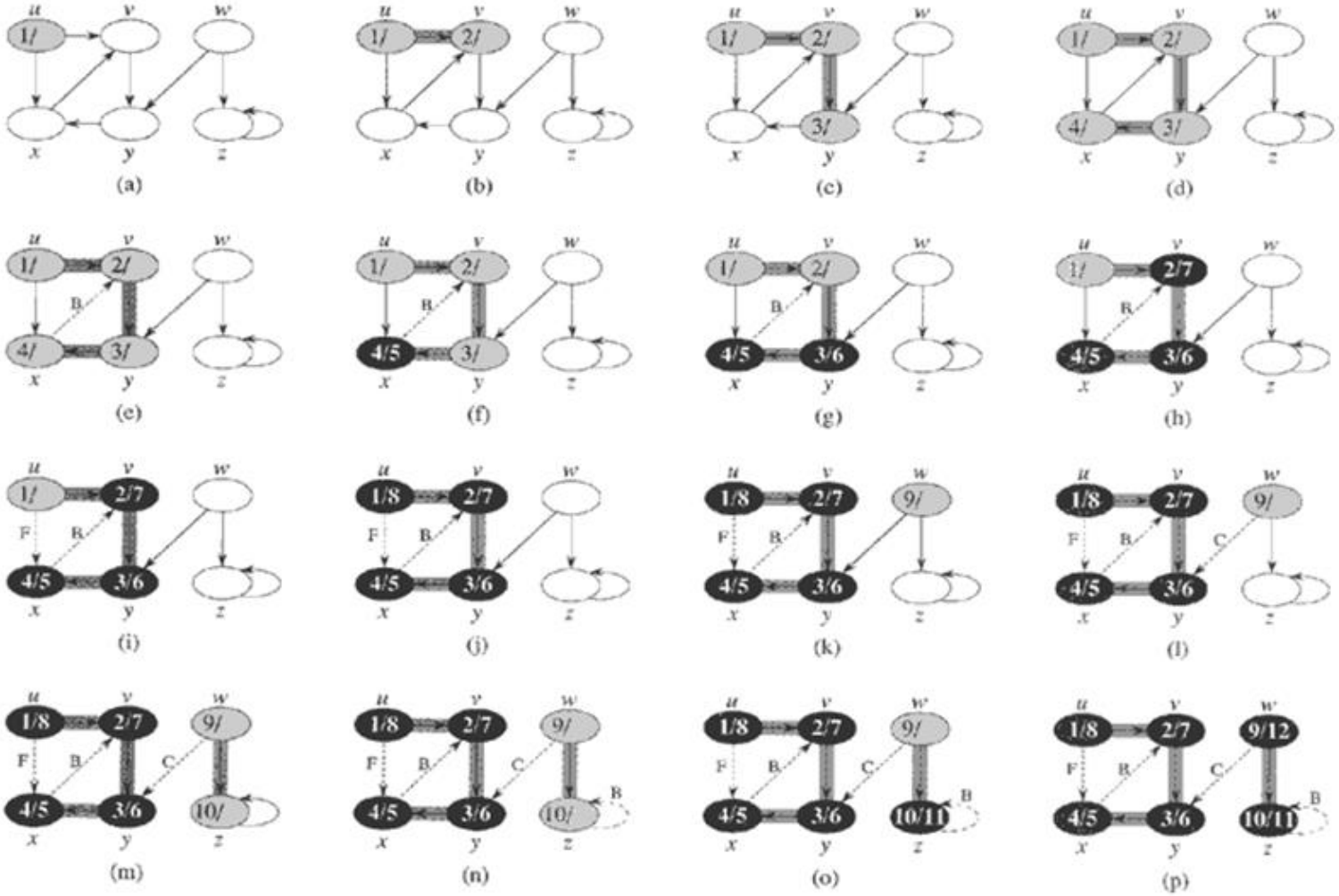
```
DFS(G)
1  for each vertex u ∈ V[G]
2       do color[u] ← WHITE
3          π[u] ← NIL
4  time ← 0
5  for each vertex u ∈ V[G]
6       do if color[u] = WHITE
7             then DFS-VISIT(u)

DFS-VISIT(u)
1  color[u] ← GRAY          ▷ White vertex u has just been discovered.
2  time ← time + 1
3  d[u] ← time
4  for each v ∈ Adj[u]      ▷ Explore edge (u, v).
5       do if color[v] = WHITE
6             then π[v] ← u
7                  DFS-VISIT(v)
8  color[u] ← BLACK         ▷ Blacken u; it is finished.
9  f[u] ← time ← time + 1
```

running time: Θ(V+E)
- 초기화: vertex 수에 비례 Θ (V)
- loop: edge 수에 비례 Θ (E)

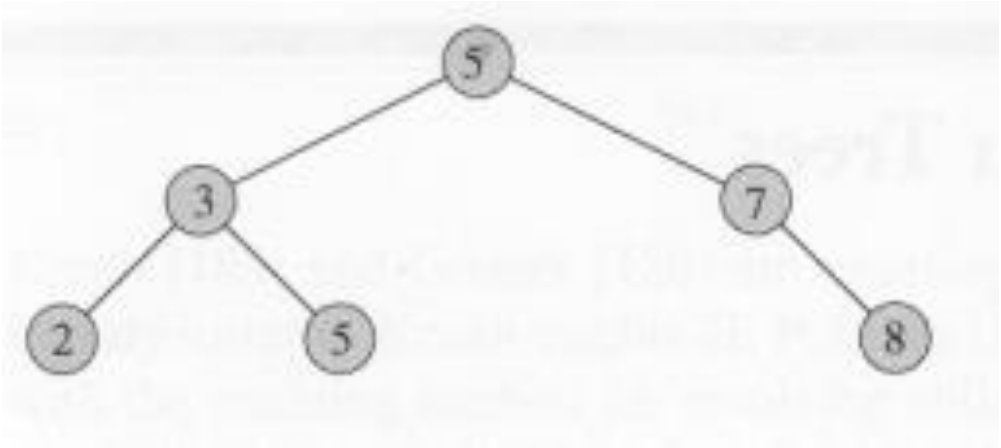
Depth-First Search

- Example



Depth-First Search

(Q)

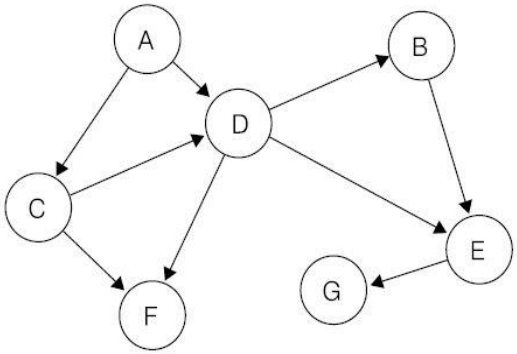


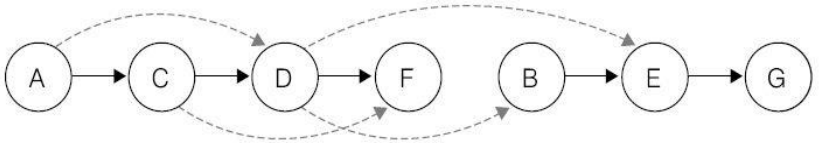
Topological Sort

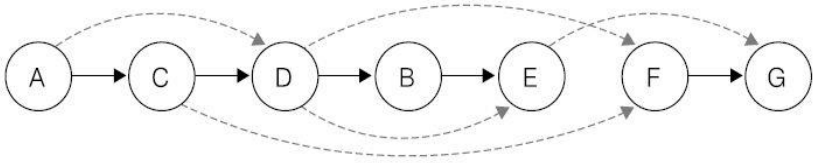
- Topological Sort (위상 정렬)
  - directed acyclic graph (dag) 의 모든 vertex 를 linear ordering
  - AOV (activity on vertex) network 에서 먼저 수행해야 될 공정부터 시작해서 일렬로 모든 공정을 나열

    Vertex : process

    Edge: process 간의 sequence







Topological Sort

- Algorithm
  1 DFS(G) 수행
  2 vertex 의 linked-list 생성(finish time 역순으로)
  3 vertex 의 linked-list return

```
TOPOLOGICAL-SORT(G)
1  call DFS(G) to compute finishing times f[v] for each vertex v
2  as each vertex is finished, insert it onto the front of a linked list
3  return the linked list of vertices
```

Topological Sort

- Example