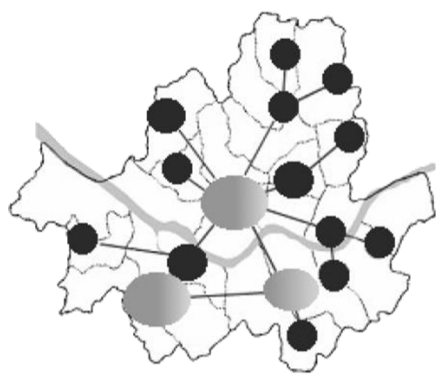
7. 그래프 알고리즘 1

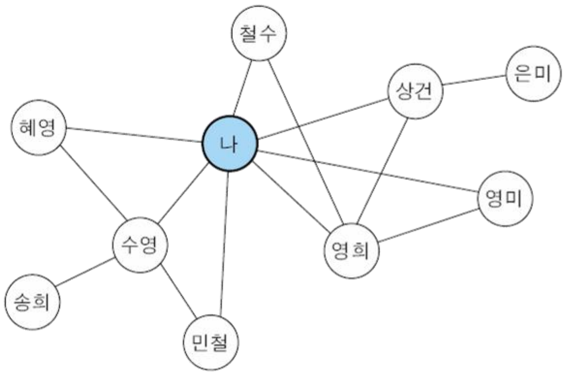
한국외국어대학교
고 석 훈

목차

7.1 그래프(Graph)

- 그래프(graph)
  - 선형 자료구조나 트리 자료구조로 표현하기 어려운 多:多의 관계를 가지는 원소들을 표현하기 위한 자료구조
- 그래프 G
  - 객체를 나타내는 정점(vertex)과 객체를 연결하는 간선(edge)의 집합
  - G = (V, E), V는 정점들의 집합, E는 간선들의 집합
- 그래프 예
  - 버스 노선도
  - 인맥도



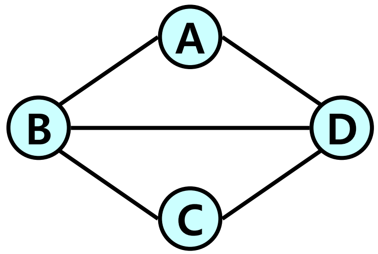


그래프의 종류

- 무방향 그래프(undirected graph)
  - 두 정점을 연결하는 간선의 방향이 없는 그래프
  - 정점 v_i와 정점 v_j을 연결하는 간선을 (v_i, v_j)로 표현
    - (v_i, v_j) 와 (v_j, v_i) 는 같은 간선을 나타낸다.
  - 무방향 그래프의 예

V(G1) = { A, B, C, D }

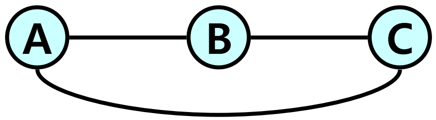
E(G1) = { (A,B), (A,D), (B,C), (B,D), (C,D) }



G1

V(G2) = { A, B, C }

E(G2) = { (A,B), (A,C), (B,C) }

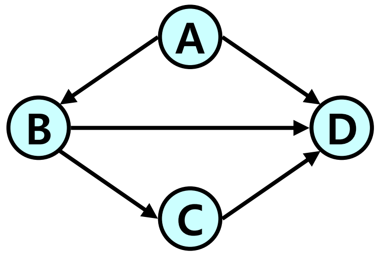


G2

- 방향 그래프(directed graph), 다이그래프(digraph)
  - 간선이 방향을 가지고 있는 그래프
  - 정점 v_i에서 정점 v_j를 연결하는 간선 $v_i \rightarrow v_j$를 $< v_i, v_j >$로 표현
    - $< v_i, v_j >$와 $< v_j, v_i >$는 서로 다른 간선이다.
  - 방향 그래프의 예

V(G3) = { A, B, C, D }

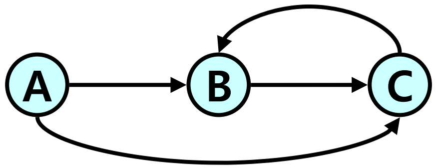
E(G3) = {<A,B>, <A,D>, <B,C>, <B,D>, <C,D>}



G3

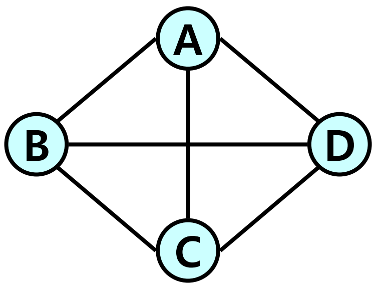
V(G4) = { A, B, C }

E(G4) = {<A,B>, <A,C>, <B,C>, <C,B>}

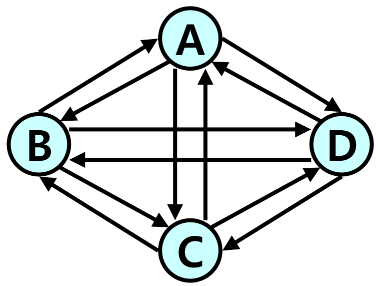


G4

- 완전 그래프(complete graph)
  - 각 정점에서 다른 모든 정점을 연결하여 가능한 최대의 간선 수를 가진 그래프
    - 정점이 n개인 무방향 그래프의 최대의 간선 수 = n(n-1)/2개
    - 정점이 n개인 방향 그래프의 최대 간선 수 = n(n-1)개
  - 완전 그래프의 예

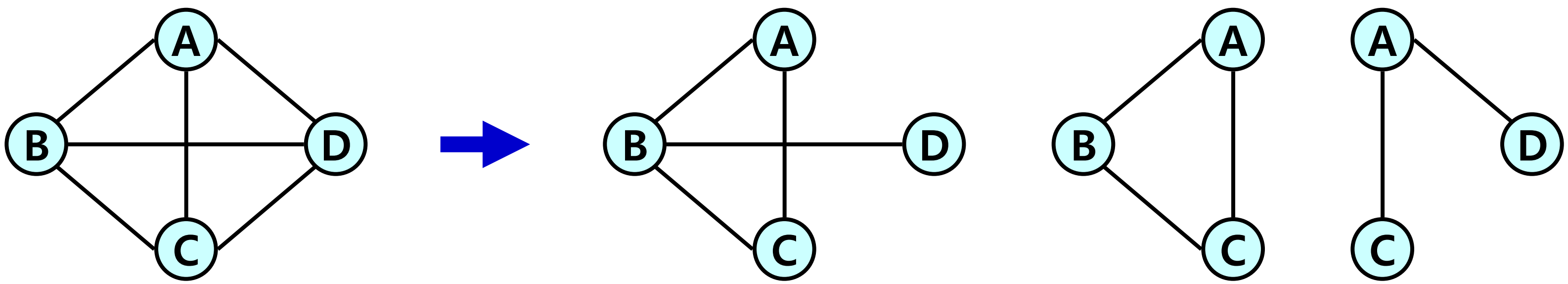


G5



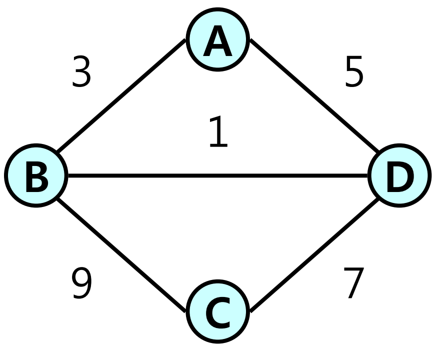
G6

- 부분 그래프(subgraph)
  - 원래 그래프에서 일부의 정점이나 간선을 제외하여 만든 그래프
  - 그래프 G와 부분 그래프 G'의 관계
    - V(G')⊆V(G), E(G')⊆E(G)
  - 그래프 G7에 대한 부분 그래프의 예

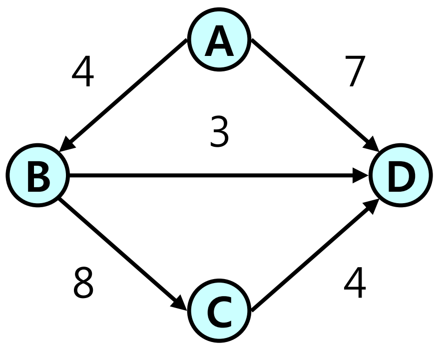


G7 G7'

- 가중 그래프(weight graph)
  - 정점을 연결하는 간선에 가중치(weight)를 할당한 그래프



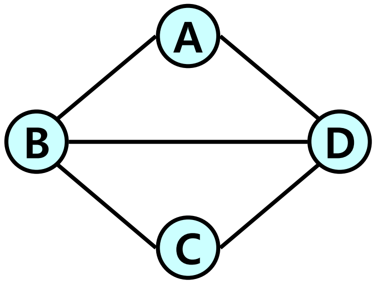
G8



G9

그래프 용어

- 인접(adjacent)과 부속(incident)
  - 그래프에서 두 정점 v_i와 v_j를 연결하는 간선 (v_i, v_j)가 있을 때, 두 정점 v_i와 v_j를 인접(adjacent)되어 있다고 하고, 간선 (v_i, v_j)는 정점 v_i와 v_j에 부속(incident)되어있다고 한다.
  - 그래프G1에서 정점 A와 인접한 정점은 B와 D 이다.
  - 정점 A에 부속되어있는 간선은 (A, B)와 (A, D) 이다.

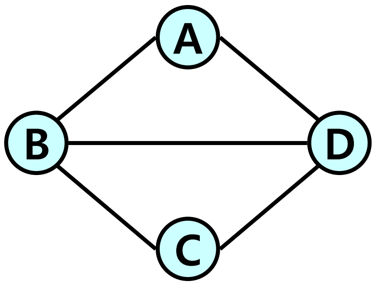


G1

- 차수(degree)
  - 무방향 그래프에서 정점에 부속되어있는 간선의 수
  - 그래프 G1에서 정점 A의 차수는 2, 정점 B의 차수는 3
  - 방향 그래프의 정점의 차수 = 진입차수 + 진출차수
    - 진입차수(in-degree) : 정점으로 들어오는 간선의 수
    - 진출차수(out-degree) : 정점에서 나가는 간선의 수
  - 방향 그래프 G3의 정점 B의 진입차수는 1, 진출차수는 2, 전체 차수는 3

G1 G3

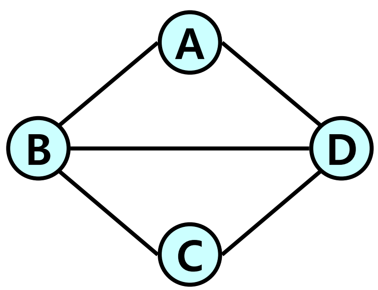
- 경로(path)
  - 그래프에서 간선을 따라 갈 수 있는 길을 순서대로 나열한 것
    즉, 정점 v_i에서 v_j까지 간선으로 연결된 정점을 순서대로 나열한 리스트
  - 그래프 G1에서 정점 A에서 정점 C까지는 A-B-C 경로,
    A-B-D-C 경로, A-D-C 경로, 그리고 A-D-B-C 경로가 있다
- 경로길이(path length)
  - 경로를 구성하는 간선의 수
  - A-B-C 경로의 길이는 2 ,
    A-B-D-C 경로의 길이는 3



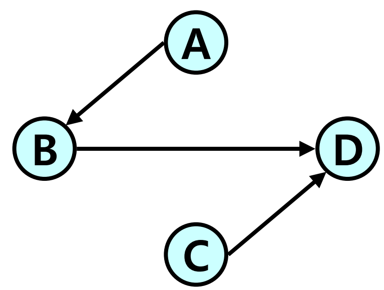
G1

- 단순경로(simple path)
  - 정점의 처음과 마지막을 제외하고 모두 다른 정점으로 구성된 경로.
    즉, 모두 다른 간선으로 구성된 경로를 의미한다.
  - 그래프 G1에서 정점 A에서 정점 C까지의 경로 A-B-C는 단순경로이고,
    경로 A-B-D-A-B-C는 단순경로가 아니다.

- 사이클(cycle)
  - 단순경로 중에서 경로의 시작 정점과 마지막 정점이 같은 경로
  - 그래프 G1에서 경로 A-B-C-D-A와 경로 B-C-D-B는 사이클이다.
- DAG(directed acyclic graph)
  - 방향 그래프이면서 사이클이 없는 그래프

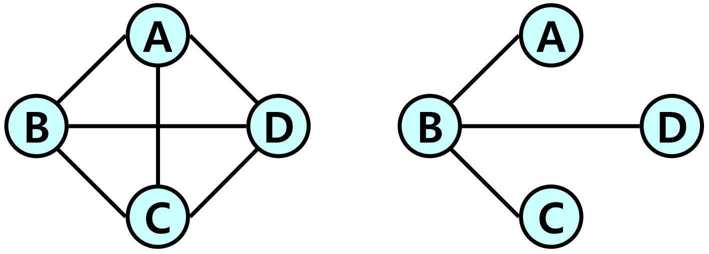


G1

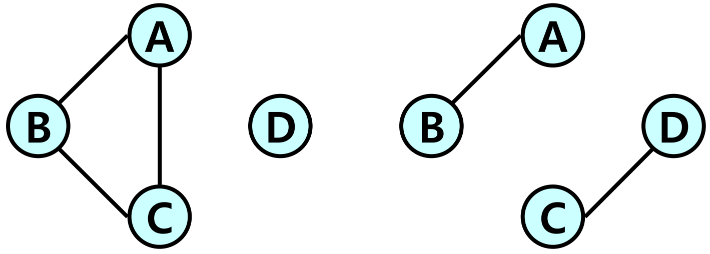


directed acyclic graph

- 연결 그래프(connected graph)
  - 서로 다른 모든 정점들 사이에 경로가 있는 그래프
  - 그래프에서 두 정점 v_i에서 v_j까지의 경로가 있으면 정점 v_i와 v_j가 연결(connected)되었다고 한다.
  - 트리는 사이클이 없는 연결 그래프이다.
- 단절 그래프(disconnected graph)
  - 연결되지 않은 정점이 있는 그래프



connected graph



disconnected graph

그래프 추상 데이터 타입(ADT)

이 름 : Graph

데이터 : 공백이 아닌 정점의 집합과 간선의 집합 (각 간선은 정점의 쌍)

연 산 : $g \in$ Graph; $v, v_1, v_2 \in V$;

initGraph(g) ::= 그래프 g를 공백 그래프로 초기화

isEmpty(g) ::= 그래프 g가 공백 그래프인지 검사

insertVertex(g, v) ::= 그래프 g에 정점 v를 삽입

insertEdge(g, v_1, v_2) ::= 그래프 g에 간선 (v_1, v_2)를 삽입

deleteVertex(g, v) ::= 정점 v와 그에 부속된 모든 간선을 삭제

deleteEdge(g, v_1, v_2) ::= 그래프 g에서 간선 (v_1, v_2)를 삭제

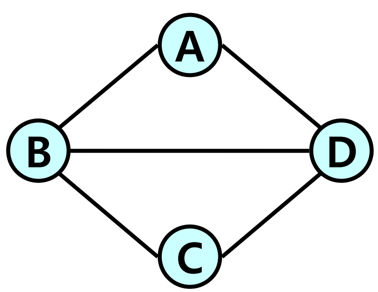
adjacent(g, v) ::= 정점 v에 인접한 모든 정점을 반환

7.2 그래프 구현

- 배열로 구현
- 연결 리스트로 구현

배열로 그래프 구현

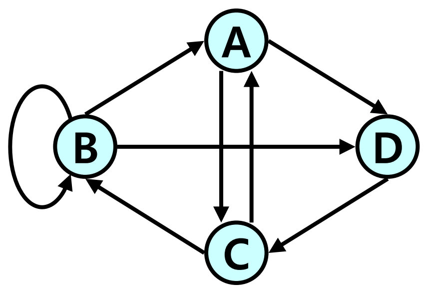
- 인접 행렬(adjacent matrix)
  - 행렬에 대한 2차원 배열을 사용하는 순차 자료구조 방법
- 무방향 그래프의 행렬 구현
  - 그래프의 두 정점을 연결한 간선의 유무를 행렬로 저장
    - n개의 정점을 가진 그래프 : n x n 정방행렬
    - 행렬의 행 번호와 열 번호 : 그래프의 정점
    - 행렬 값 : 두 정점이 인접되어있으면 1, 인접되어있지 않으면 0



G1

	A	B	C	D
A		1		1
B	1		1	1
C		1		1
D	1	1	1	

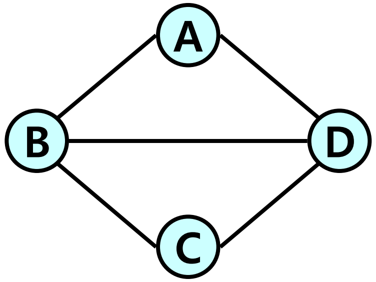
- 방향 그래프의 행렬 구현
  - n개의 정점을 가진 그래프 : n x n 정방행렬
    - 행렬의 행 번호 : 진출 정점
    - 행렬의 열 번호 : 진입 정점
    - 행렬 값 : 두 정점이 인접되어있으면 1, 인접되어있지 않으면 0



G2

	A	B	C	D
A			1	1
B	1	1		1
C	1	1		
D			1	

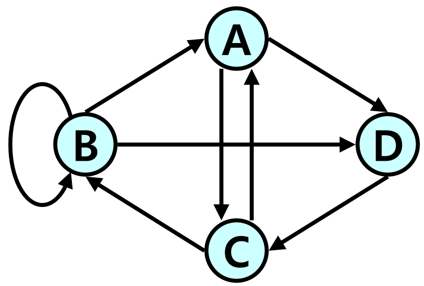
- 무방향 그래프의 인접 행렬
  - 행 i의 합 = 열 i의 합
    = 정점 i의 차수



G1

	A	B	C	D	
A		1		1	
B	1		1	1	3
C		1		1	
D	1	1	1		
		3			

- 방향 그래프의 인접 행렬
  - 행 i의 합 =
    정점 i의 진출차수
  - 열 i의 합 =
    정점 i의 진입차수



G2

	A	B	C	D	
A			1	1	
B	1	1		1	3
C	1	1			
D			1		
		2			

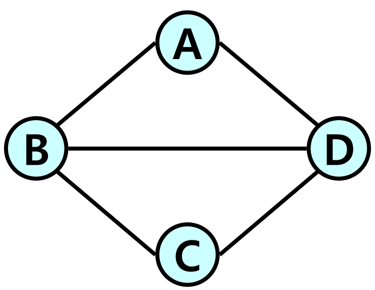
- 인접 행렬 표현의 단점
  - n개의 정점을 가지는 그래프를 항상 n x n개의 메모리 사용
  - 정점의 개수에 비해서 간선의 개수가 적은
    희소 그래프에 대한 인접 행렬은 희소 행렬이 되므로
    메모리의 낭비 발생

연결 리스트로 그래프 구현

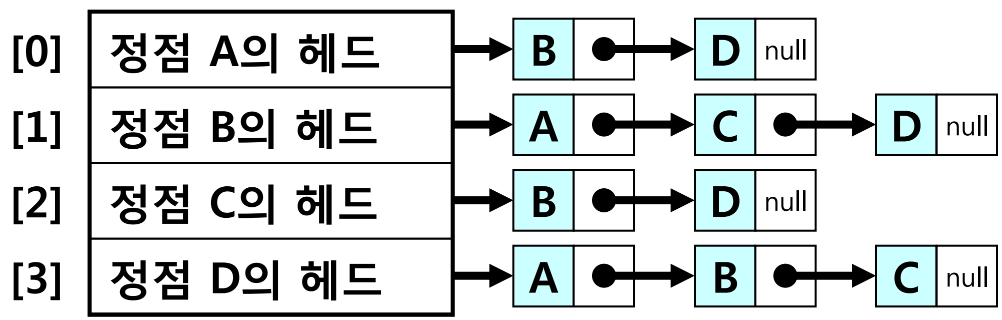
- 인접 리스트(adjacent list)
  - 각 정점에 대한 인접 정점들을 연결하여 만든 단순 연결 리스트
  - 각 정점의 차수만큼 노드를 연결
    - 리스트 내의 노드들은 인접 정점에 대해서 오름차순으로 연결
  - 인접 리스트의 각 노드
    - 정점을 저장하는 필드와 다음 인접 정점을 연결하는 링크 필드로 구성
  - 정점의 헤드 노드
    - 정점에 대한 리스트의 시작을 표현

연결 리스트로 무방향 그래프 구현

- n개의 정점과 e개의 간선을 가진 무방향 그래프의 인접 리스트
  - 헤드 노드 배열의 크기 : n
  - 연결하는 노드의 수 : 2e
  - 각 정점의 헤드에 연결된 노드의 수 : 정점의 차수

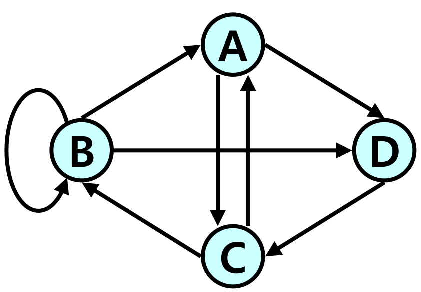


G1

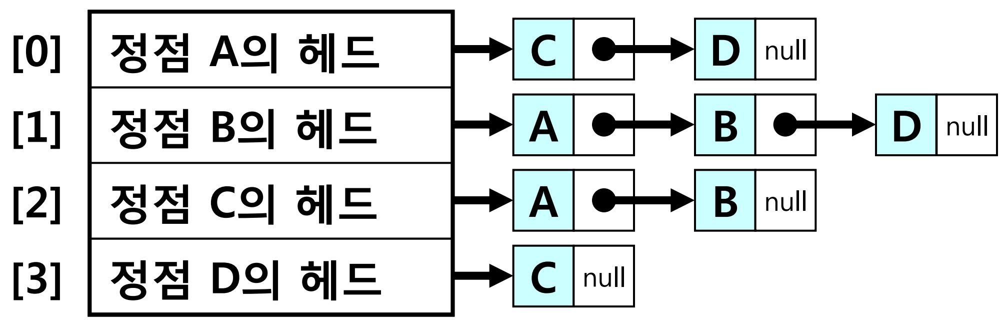


● 연결 리스트로 방향 그래프 구현

- n개의 정점과 e개의 간선을 가진 방향 그래프의 인접 리스트
  - 헤드 노드 배열의 크기 : n
  - 연결하는 노드의 수 : e
  - 각 정점의 헤드에 연결된 노드의 수 : 정점의 진출 차수



G2



7.3 그래프 순회(Graph Traversal)

- 그래프 순회(graph traversal), 그래프 탐색(graph search)
  - 하나의 정점에서 시작하여 그래프에 있는 모든 정점을 한번씩 방문하여 처리하는 연산

- 그래프 탐색방법
  - 깊이 우선 탐색(depth first search, DFS)
  - 너비 우선 탐색(breadth first search, BFS)

깊이 우선 탐색(DFS)

- 깊이 우선 탐색(depth first search, DFS)
  - 시작 정점에서 한 방향으로 갈 수 있는 경로가 있는 곳까지 깊이 탐색을 하다가 더 이상 갈 곳이 없게 되면, 가장 마지막에 만났던 갈림길 간선이 있는 정점으로 되돌아와서, 다음 갈림길 방향으로 깊이 탐색을 계속 반복하여 모든 정점을 방문하는 순회방법
  - 갈 곳이 없는 경우, 가장 최근에 만났던 갈림길 간선의 정점으로 되돌아가서 깊이 우선 탐색을 반복해야 하므로
    후입선출(LIFO) 구조의 스택(stack) 사용

DFS 알고리즘 1: Stack과 Loop

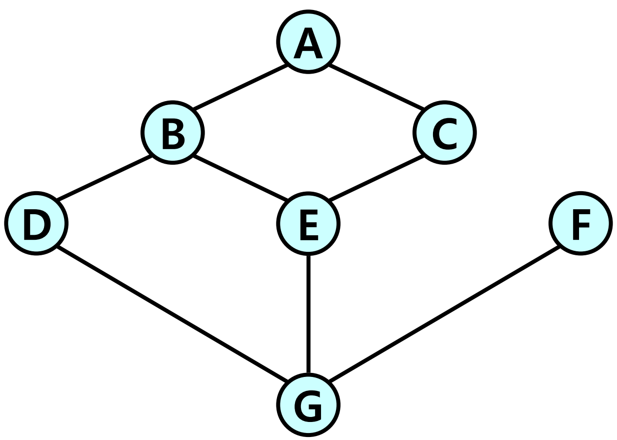
```
DFS1(v)
    for (i ← 0; i < n; i++) do
        visited[i] ← false;
    push(stack, v);
    while (not isEmpty(stack)) do {
        v ← pop(stack);
        if (visited[v] = false) then {
            visited[v] ← true;
            v 방문;        // print v값
            for (v와 인접한 모든 정점 w)      // 단, w를 역순으로 push
                push(stack, w);
        }
    }
end DFS1()
```

DFS 알고리즘 2: Recursive

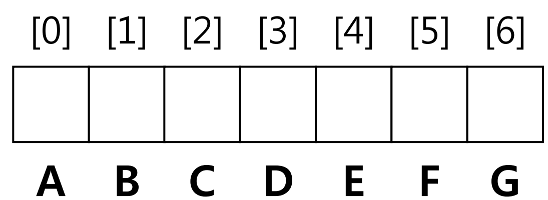
```
dfsCore(v)
    visited[v] ← true;
    v 방문;        // print v값
    for (v와 인접한 모든 정점 w) do {
        if (visited[w] = false)
            dfsCore(w);
    }
end dfsCore()

DFS2(v)
    for (i ← 0; i < n; i++) do
        visited[i] ← false;
    dfsCore(v);
end DFS2()
```

깊이 우선 탐색(DFS) 예

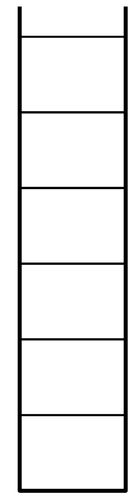
- 초기화
  - 배열 visited를 False로 초기화하고, 공백 스택을 생성한다.



G9

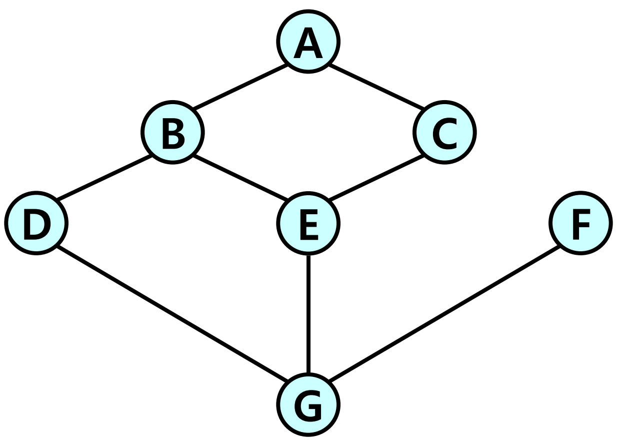


visited



stack

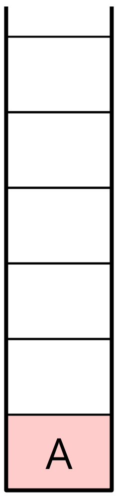
- 시작 정점 A를 스택에 push



G9

[0]	[1]	[2]	[3]	[4]	[5]	[6]
A	**B**	**C**	**D**	**E**	**F**	**G**

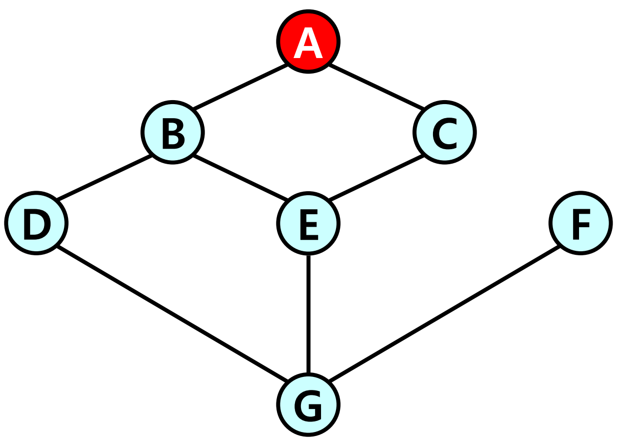
visited



stack

DFS:

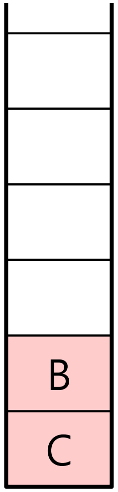
- 스택이 비어있지 않으므로 스택에서 정점 A를 pop한다.
- 정점 A를 방문하지 않았으므로
  - 정점 A를 방문하고, 인접정점 B, C를 역순으로 스택에 push



G9

[0]	[1]	[2]	[3]	[4]	[5]	[6]
T						
A	**B**	**C**	**D**	**E**	**F**	**G**

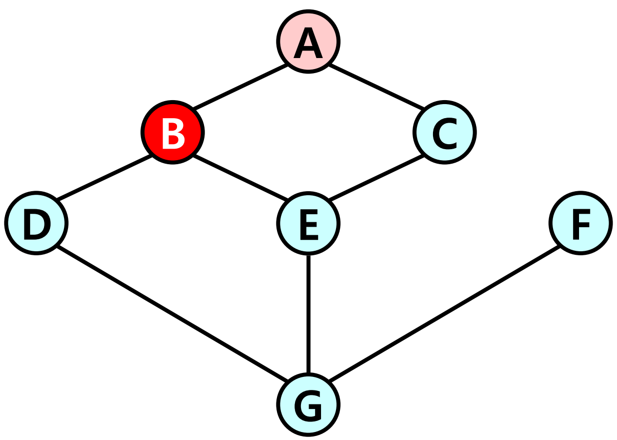
visited



stack

DFS: A

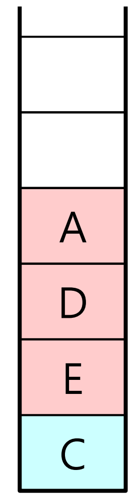
- 스택이 비어 있지 않으므로 스택에서 정점 B를 pop한다.
  - 정점 B를 방문하지 않았으므로 정점 B를 방문하고,
    인접 정점 A, D, E를 역순으로 스택에 push 한다.



G9

[0]	[1]	[2]	[3]	[4]	[5]	[6]
T	T					
A	**B**	**C**	**D**	**E**	**F**	**G**

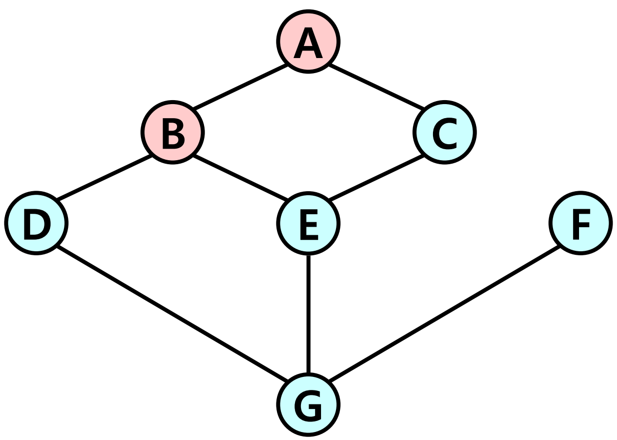
visited



stack

DFS: A – B

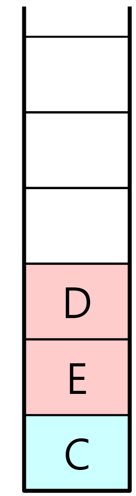
- 스택이 비어있지 않으므로 스택에서 정점 A를 pop한다.
  - 정점 A는 이미 방문하였으므로 다시 스택을 확인한다.



G9

[0]	[1]	[2]	[3]	[4]	[5]	[6]
T	T					
A	**B**	**C**	**D**	**E**	**F**	**G**

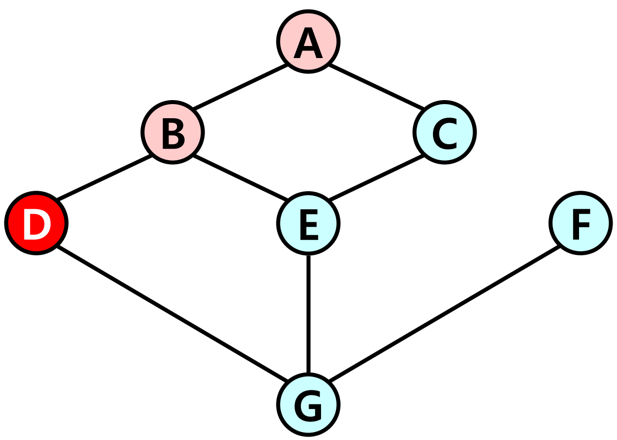
visited



stack

DFS: A – B

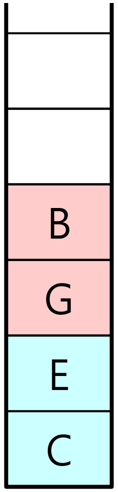
- 스택이 비어있지 않으므로 스택에서 정점 D를 pop한다.
  - 정점 D를 방문하지 않았으므로 정점 D를 방문하고, 인접정점 B, G를 역순으로 스택에 push 한다.



G9

[0]	[1]	[2]	[3]	[4]	[5]	[6]
T	T		T			
A	**B**	**C**	**D**	**E**	**F**	**G**

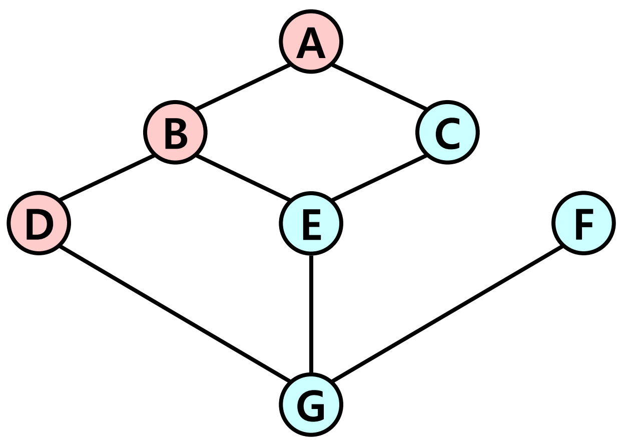
visited



stack

DFS: A – B – D

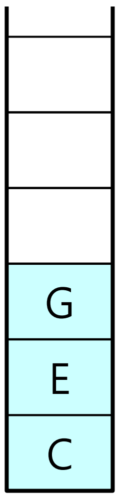
- 스택이 비어있지 않으므로 스택에서 정점 B를 pop한다.
  - 정점 B는 이미 방문하였으므로 다시 스택을 확인한다.



G9

[0]	[1]	[2]	[3]	[4]	[5]	[6]
T	T		T			
A	**B**	**C**	**D**	**E**	**F**	**G**

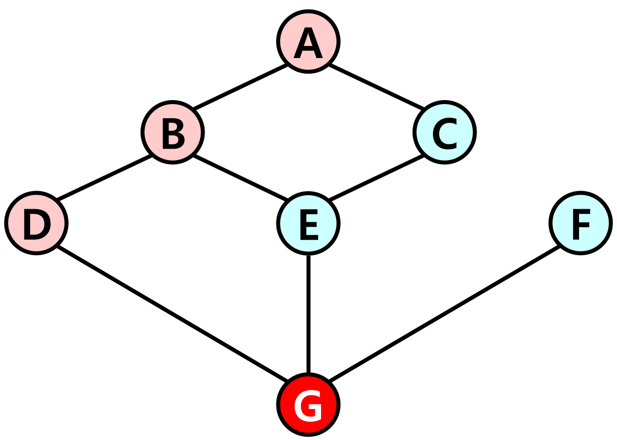
visited



stack

DFS: A – B – D

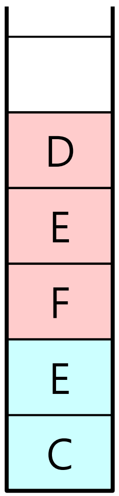
- 스택이 비어있지 않으므로 스택에서 정점 G를 pop한다.
  - 정점 G를 방문하지 않았으므로 정점 G를 방문하고, 인접정점 D, E, F를 역순으로 스택에 push 한다.



G9

[0]	[1]	[2]	[3]	[4]	[5]	[6]
T	T		T			T
A	**B**	**C**	**D**	**E**	**F**	**G**

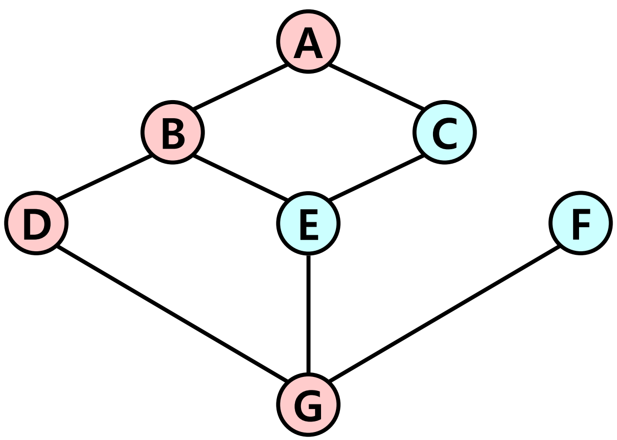
visited



stack

DFS: A – B – D – G

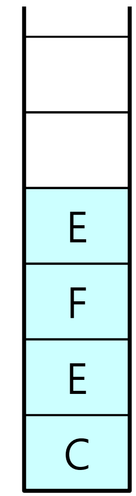
- 스택이 비어있지 않으므로 스택에서 정점 D를 pop한다.
  - 정점 D는 이미 방문하였으므로 다시 스택을 확인한다.



G9

[0]	[1]	[2]	[3]	[4]	[5]	[6]
T	T		T			T
A	**B**	**C**	**D**	**E**	**F**	**G**

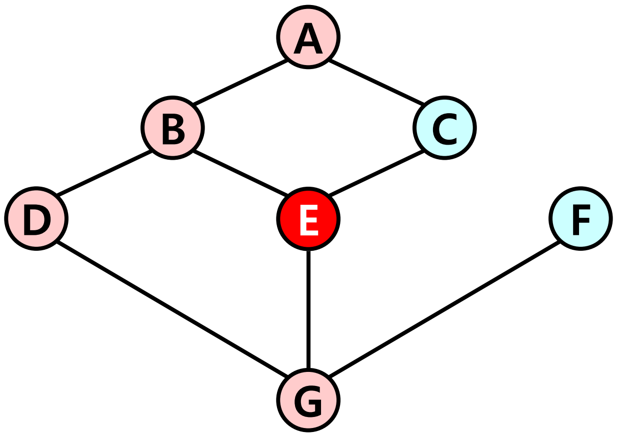
visited



stack

DFS: A – B – D – G

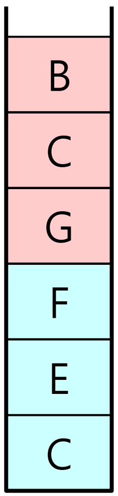
- 스택이 비어있지 않으므로 스택에서 정점 E를 pop한다.
  - 정점 E를 방문하지 않았으므로 정점 E를 방문하고,
    인접정점 B, C, G를 역순으로 스택에 push 한다.



G9

[0]	[1]	[2]	[3]	[4]	[5]	[6]
T	T		T	**T**		T
A	**B**	**C**	**D**	**E**	**F**	**G**

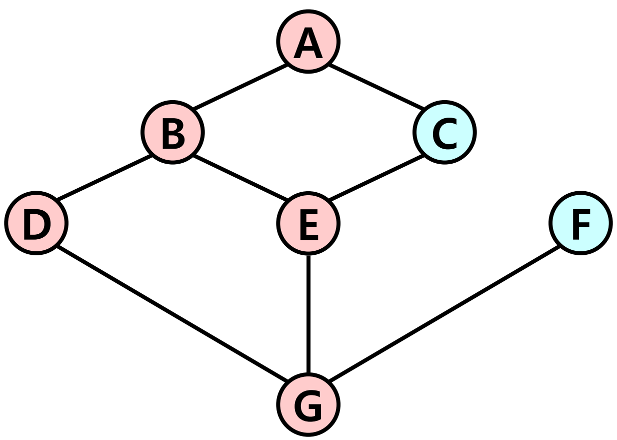
visited



stack

DFS: A – B – D – G – E

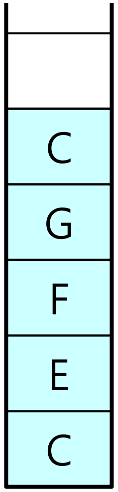
- 스택이 비어있지 않으므로 스택에서 정점 B를 pop한다.
  - 정점 B는 이미 방문하였으므로 다시 스택을 확인한다.



G9

[0]	[1]	[2]	[3]	[4]	[5]	[6]
T	T		T	T		T
A	**B**	**C**	**D**	**E**	**F**	**G**

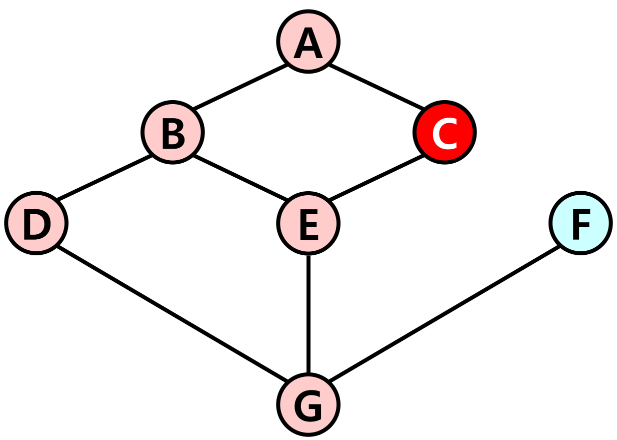
visited



stack

DFS: A – B – D – G – E

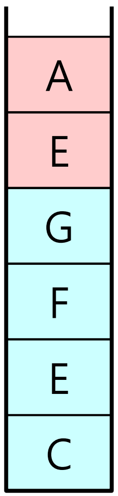
- 스택이 비어있지 않으므로 스택에서 정점 C를 pop한다.
  - 정점 C를 방문하지 않았으므로 정점 C를 방문하고,
    인접정점 A, E를 역순으로 스택에 push 한다.



G9

[0]	[1]	[2]	[3]	[4]	[5]	[6]
T	T	T	T	T		T
A	B	C	D	E	F	G

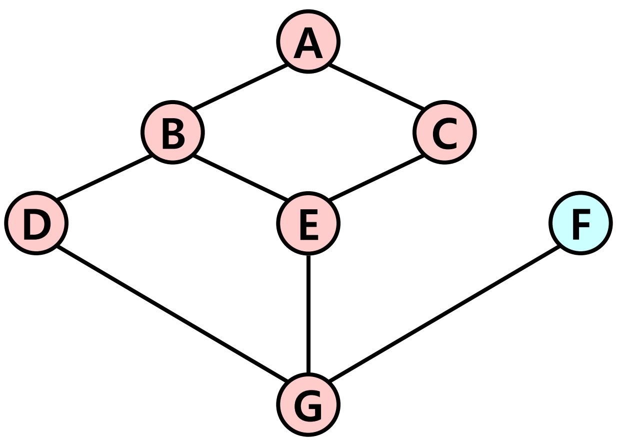
visited



stack

DFS: A – B – D – G – E – C

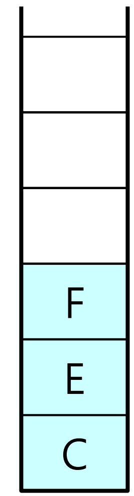
- 스택이 비어있지 않으므로 스택에서 정점 A를 pop한다.
  - 정점 A는 이미 방문하였으므로 다시 스택을 확인한다.
- 같은 식으로 정점 E, G를 pop하고 스택을 확인한다.



G9

	[0]	[1]	[2]	[3]	[4]	[5]	[6]
	T	T	T	T	T		T
	A	**B**	**C**	**D**	**E**	**F**	**G**

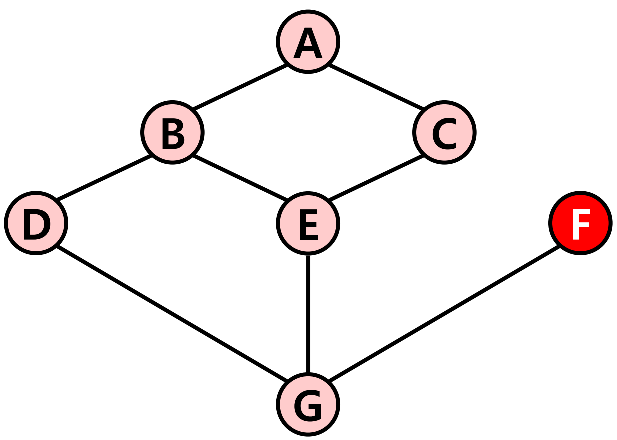
visited



stack

DFS: A – B – D – G – E – C

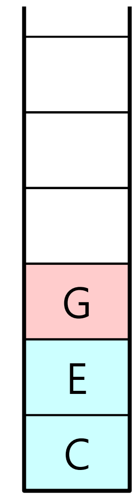
- 스택이 비어있지 않으므로 스택에서 정점 F를 pop한다.
  - 정점 F를 방문하지 않았으므로 정점 F를 방문하고, 인접정점 G를 스택에 push 한다.



G9

[0]	[1]	[2]	[3]	[4]	[5]	[6]
T	T	T	T	T	**T**	T
A	**B**	**C**	**D**	**E**	**F**	**G**

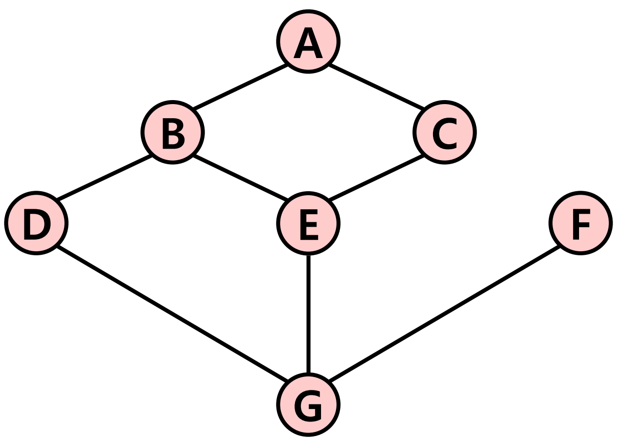
visited



stack

DFS: A – B – D – G – E – C – F

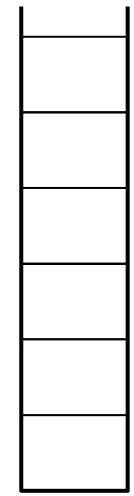
- 스택이 비어있지 않으므로 스택에서 정점 G를 pop한다.
  - 정점 G는 이미 방문하였으므로 다시 스택을 확인한다.
  - 같은 식으로 정점 E, C를 pop하고 스택을 확인한다.
  - 마지막으로 스택이 비게 되면 실행을 종료한다.



G9

[0]	[1]	[2]	[3]	[4]	[5]	[6]
T	T	T	T	T	T	T
A	B	C	D	E	F	G

visited



stack

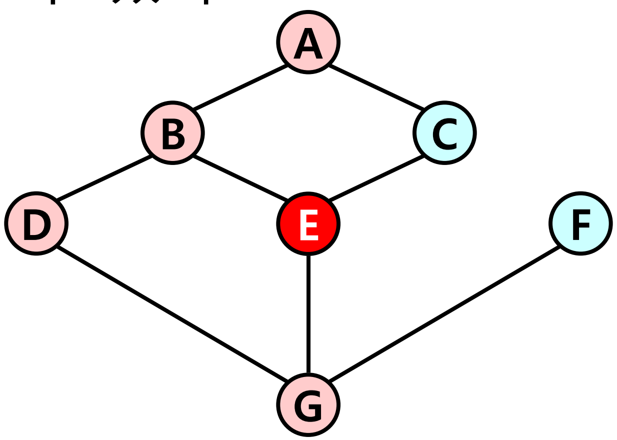
DFS: A – B – D – G – E – C – F

깊이 우선 탐색(DFS) 알고리즘 응용

- 그래프 사이클 검출?
  - 깊이 우선 탐색 알고리즘에서
    현재 방문한 정점의 인접정점 중에
    직전에 방문한 정점을 제외하고, 이미 방문한 정점이 있다면
    사이클이 발생함을 확인할 수 있다.

그래프 사이클 검출 예

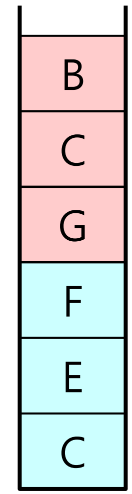
- 정점 E를 방문하고, 인접정점 B, C, G를 스택에 push하는 단계에서 직전 방문한 정점 G를 제외하고, 인접정점 B가 이미 방문한 정점이므로 B – D – G – E – B 사이클이 발생함을 확인할 수 있다.



G9

[0]	[1]	[2]	[3]	[4]	[5]	[6]
T	T		T	T		T
A	**B**	**C**	**D**	**E**	**F**	**G**

visited



stack

DFS: A – B – D – G – E – B / – C / – G

너비 우선 탐색(BFS)

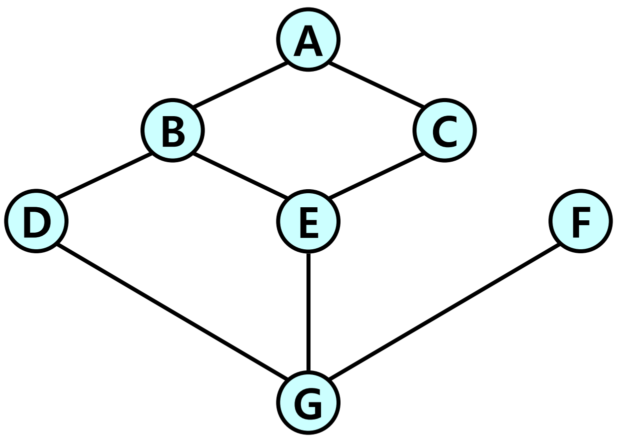
- 너비 우선 탐색(breadth first search, BFS) 순회 방법
  - 시작 정점으로부터 인접한 정점들을 모두 차례로 방문하고 나서, 방문했던 각 정점을 시작 정점으로 하여 다시 인접한 정점들을 차례로 방문한다.
  - 인접한 정점들에 대해서 차례로 다시 너비 우선 탐색을 반복해야 하므로 선입선출(FIFO) 구조를 갖는 큐(queue)를 사용한다.
  - 가까운 정점들을 먼저, 멀리 있는 정점들은 나중에 방문하는 순회 방법이다.

BFS 알고리즘

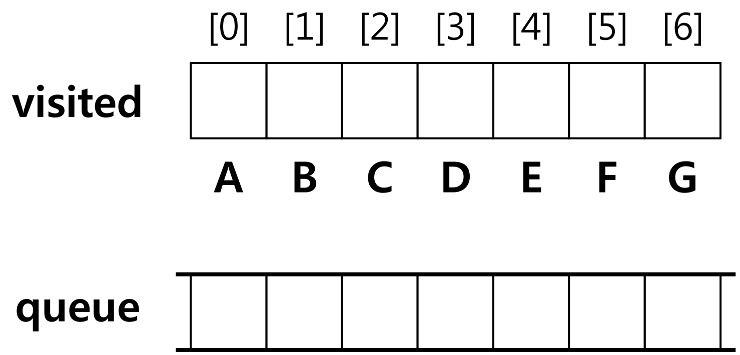
```
BFS(v)
      for (i ← 0; i < n; i++) do
            visited[i] ← false;
      visited[v] ← true;
      v 방문;          // print v값
      enqueue(Q, v);
      while (not isEmpty(Q)) do {
            v ← dequeue(Q);
            for (모든 v의 인접 정점 w) do {
                  if (visited[w] = false) {
                        visited[w] ← true;
                        w 방문;          // print w값
                        enqueue(Q, w);
                  }
            }
      }
end BFS()
```

너비 우선 탐색(BFS) 예

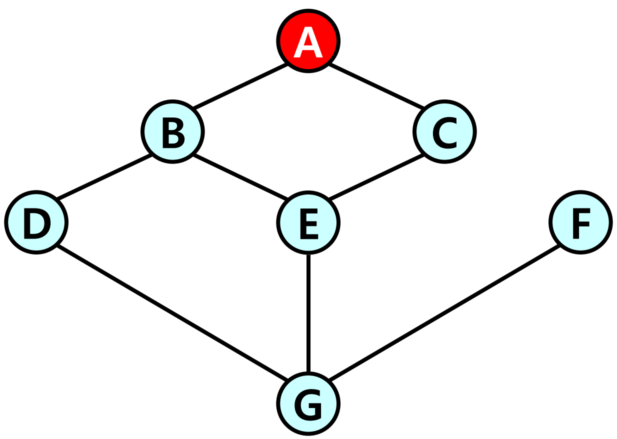
- 초기상태: 배열 visited를 False로 초기화하고, 공백 큐를 생성



G9



- 시작 정점 A를 방문하고, enqueue한다.



G9

visited

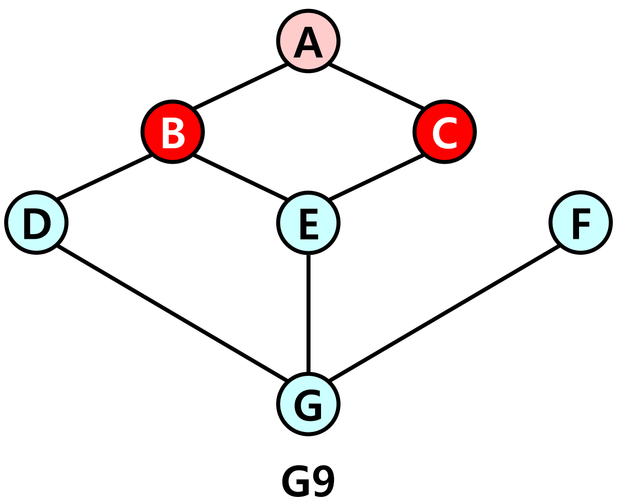
[0]	[1]	[2]	[3]	[4]	[5]	[6]
T						
A	B	C	D	E	F	G

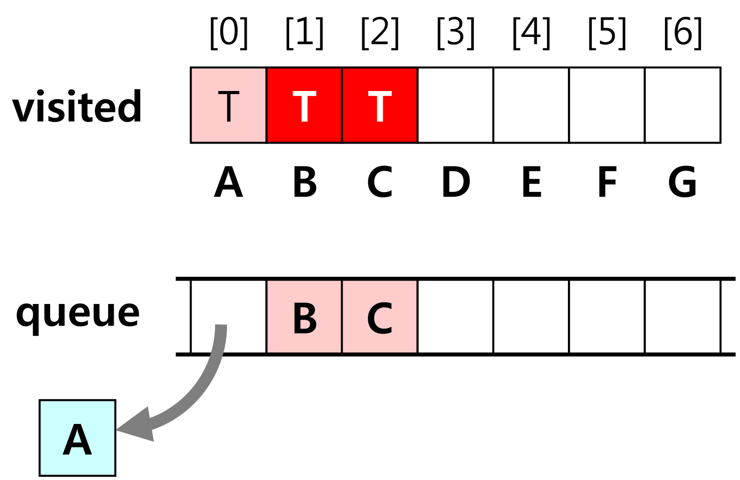
queue

A						

BFS: A

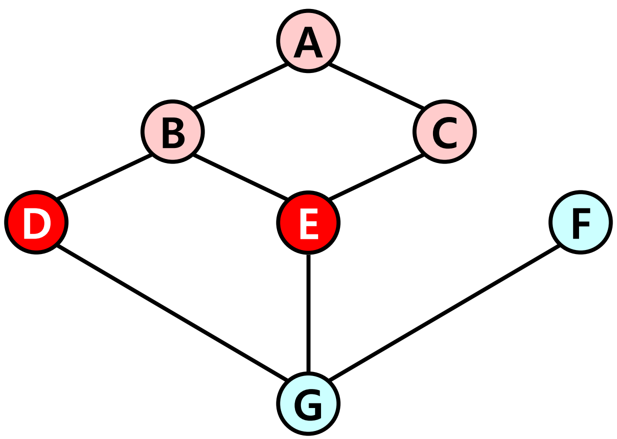
- 큐가 비어있지 않으므로 정점A를 dequeue한다.
  - 정점 A의 방문 안 한 모든 인접정점 B, C를 방문하고, enqueue한다.



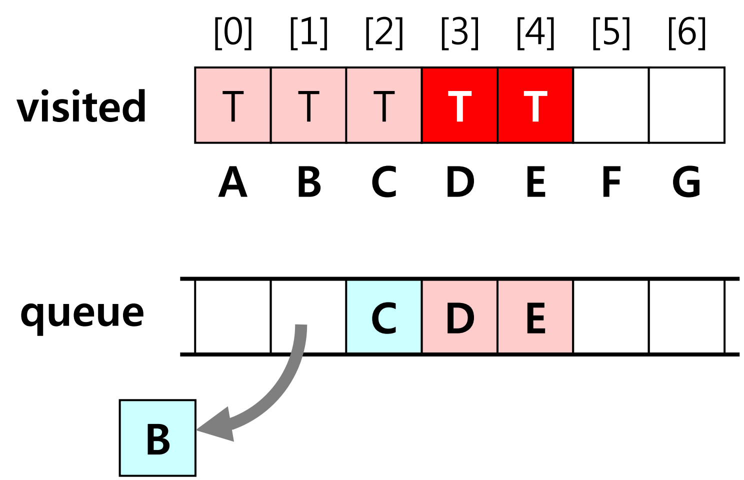


BFS: A – B – C

- 큐가 비어있지 않으므로 정점B를 dequeue한다.
  - 정점 B의 방문 안 한 모든 인접정점 D, E를 방문하고, enqueue한다.

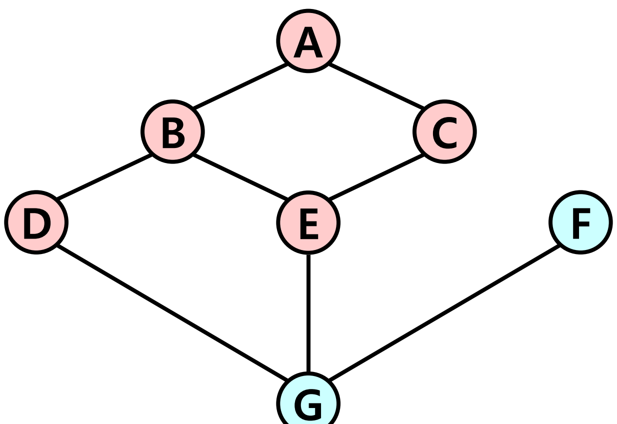


G9

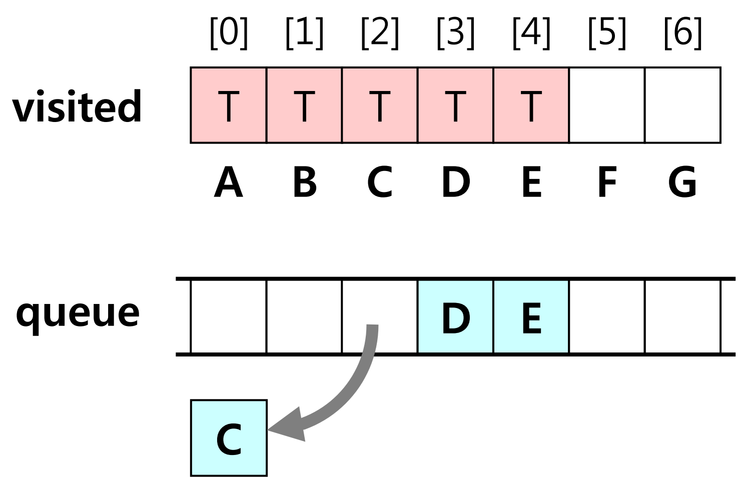


BFS: A – B – C – D – E

- 큐가 비어있지 않으므로 정점C를 dequeue한다.
  - 정점 C의 방문 안 한 인접정점은 없다.

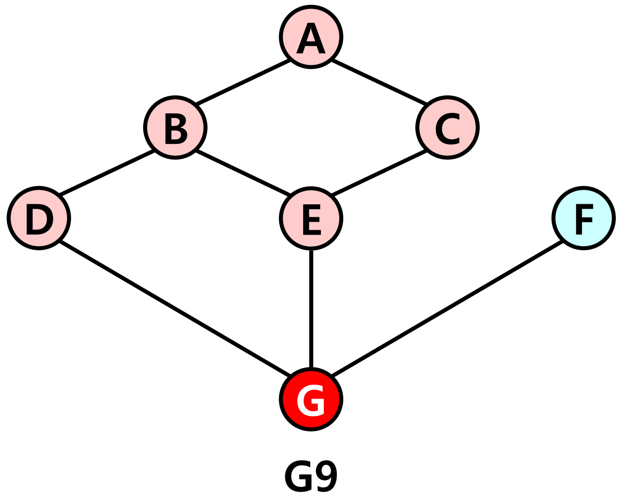


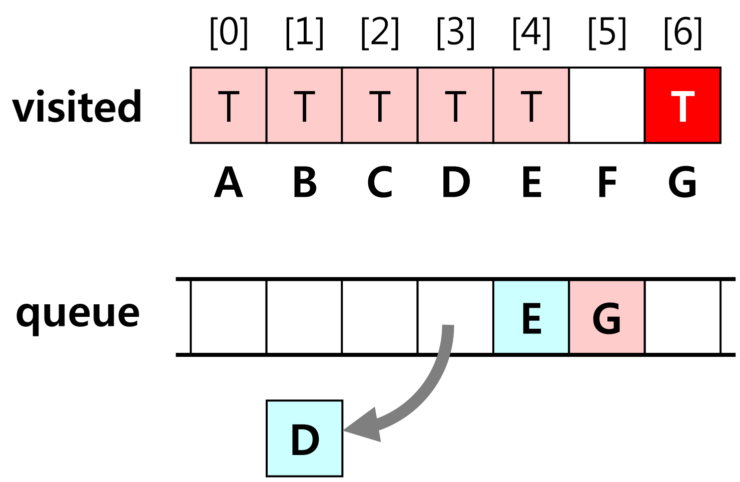
G9



BFS: A – B – C – D – E

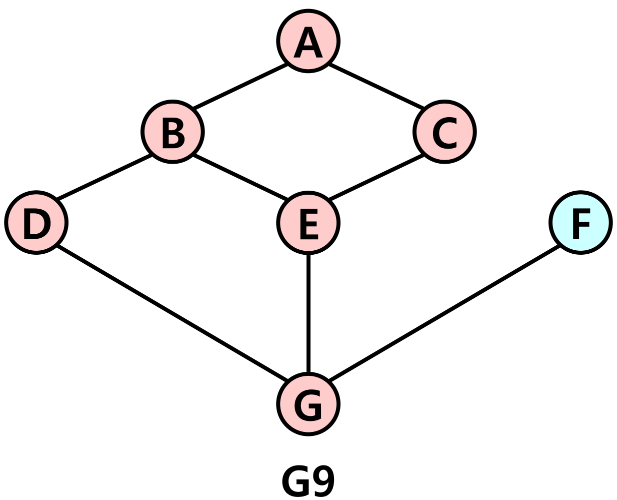
- 큐가 비어있지 않으므로 정점D를 dequeue한다.
  - 정점 D의 방문 안 한 인접정점 G를 방문하고, enqueue한다.

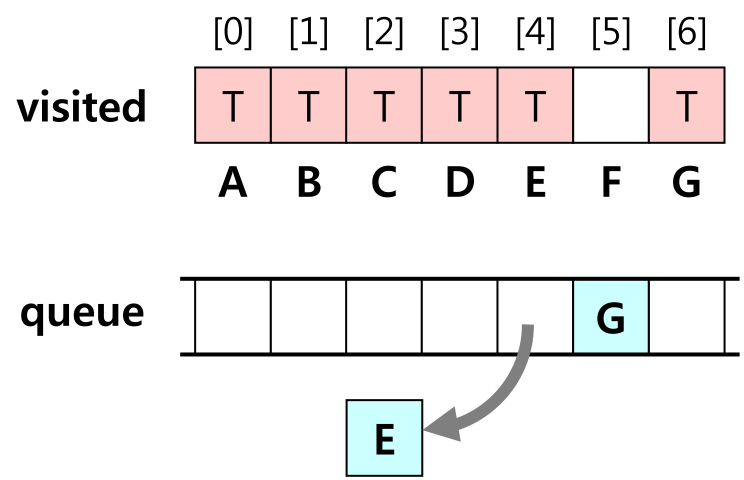




BFS: A – B – C – D – E – G

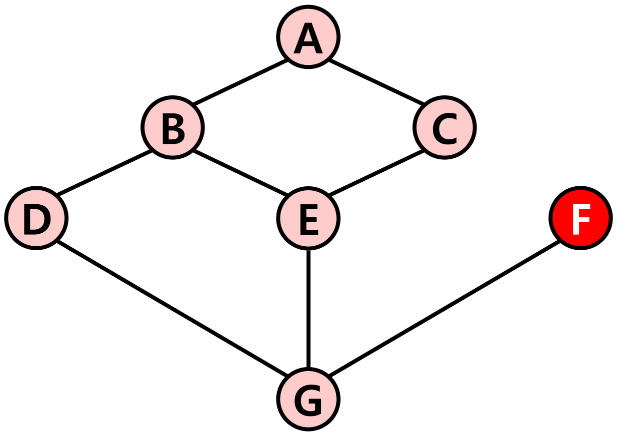
- 큐가 비어있지 않으므로 정점E를 dequeue한다.
  - 정점 E의 방문 안 한 인접정점은 없다.



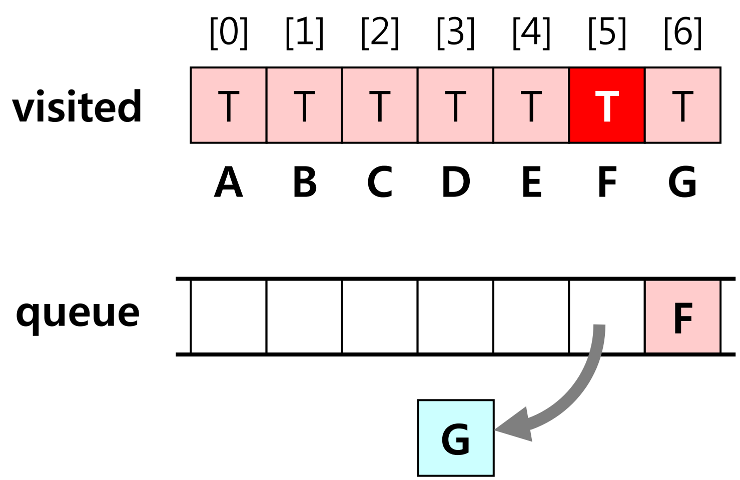


BFS: A – B – C – D – E – G

- 큐가 비어있지 않으므로 정점G를 dequeue한다.
  - 정점 G의 방문 안 한 인접정점 F를 방문하고, enqueue한다.

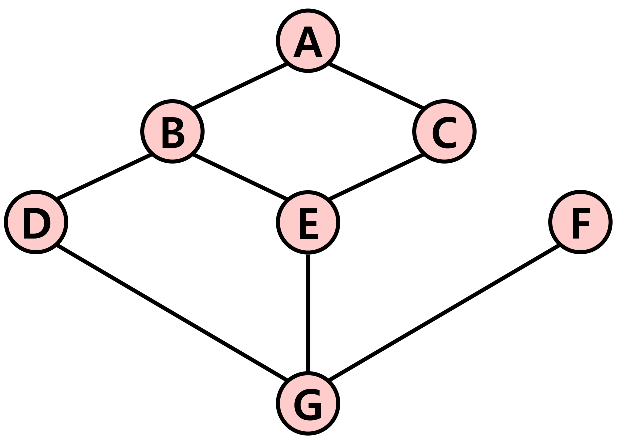


G9

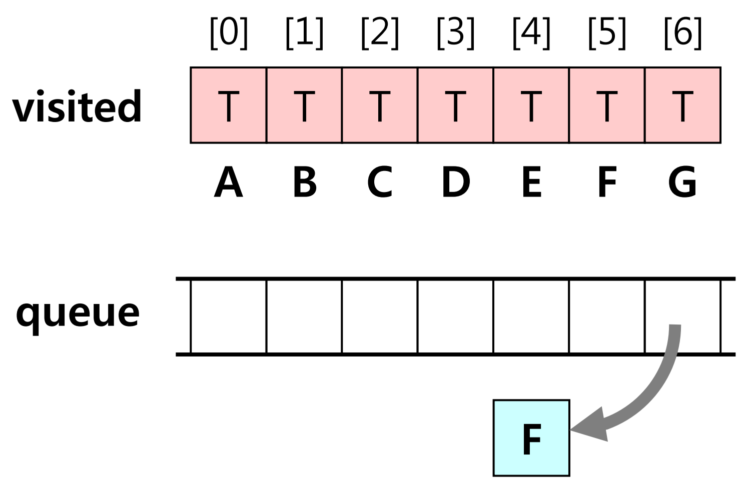


BFS: A – B – C – D – E – G – F

- 큐가 비어있지 않으므로 정점F를 dequeue한다.
  - 정점 F의 방문 안 한 인접정점은 없다.
  - 큐가 비었으므로 실행을 종료한다.



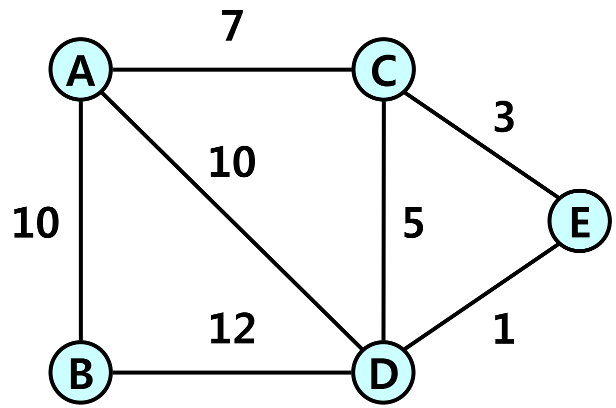
G9



BFS: A – B – C – D – E – G – F

7.4 최소 신장 트리

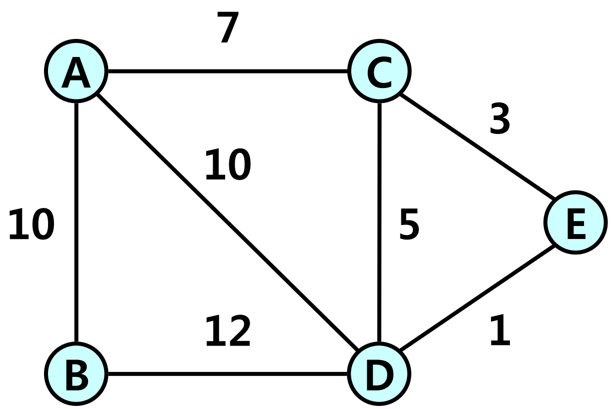
- 가중 그래프(weight graph)
  - 정점을 연결하는 간선에 가중치(weight)를 할당한 그래프



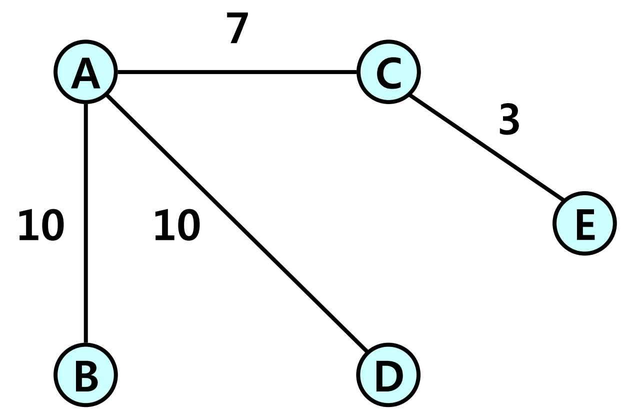
G1

신장 트리(Spanning Tree)

- 신장 트리(Spanning Tree)
  - 그래프 G에서 사이클 없이 그래프 G의 모든 정점과 그 정점을 연결하는 간선 들로 구성된 트리
  - 정점의 개수가 n개인 경우, 신장 트리의 간선은 항상 n-1개가 된다.
  - Weight(T) = T의 간선의 가중치의 합



G1

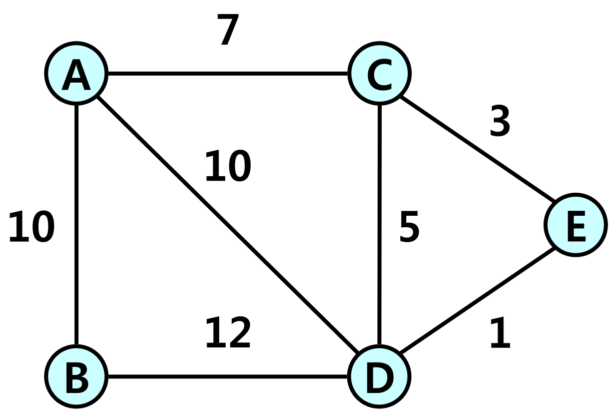


T1

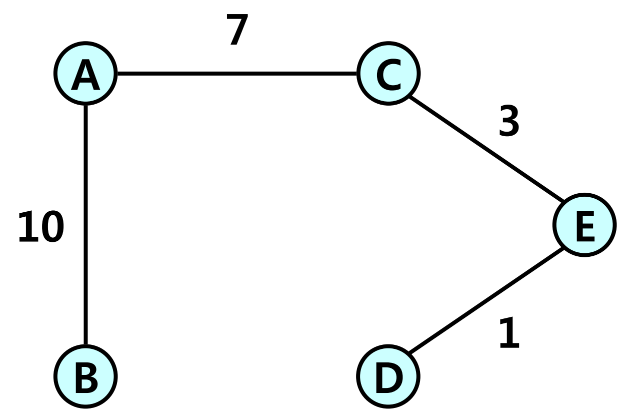
Weight(T1) = 10 + 10 + 7 + 3 = 30

최소 신장 트리(MST)

- 최소 신장 트리 (MST, Minimum Spanning Tree)
  - G의 신장 트리 T 중에서 Weight(T)가 가장 작은 신장 트리
  - 그래프 G1의 최소 신장 트리는?



G1



T2

Weight(T2) = 10 + 7 + 3 + 1 = 21

최소 신장 트리(MST) 알고리즘

- 최소 신장 트리를 찾는 그리디 알고리즘
  - Kruskal 알고리즘
  - Prim 알고리즘

Kruskal 알고리즘

- Kruskal 알고리즘

  (1) 그래프 G의 모든 간선을 가중치에 따라 오름차순으로 정리한다.

  (2) 그래프 G에 가중치가 가장 작은 간선을 삽입한다.

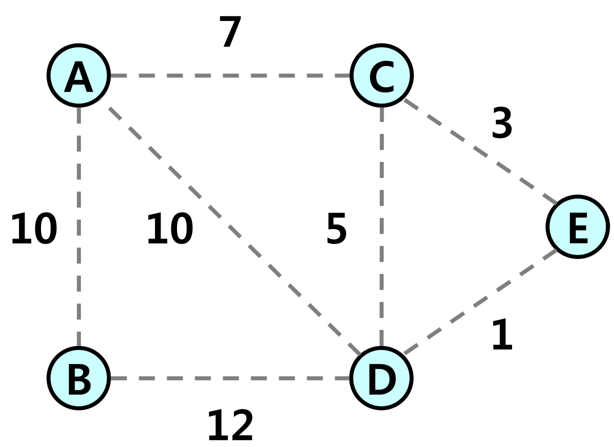
  - 이때 사이클을 형성하는 간선은 삽입할 수 없으므로
    이런 경우에는 그 다음으로 가중치가 작은 간선을 삽입한다.

  (3) 그래프 G에 n-1개의 간선을 삽입할 때까지 (2)를 반복한다.

  (4) 그래프 G의 간선이 n-1개가 되면 최소 신장 트리가 완성된다.

Kruskal 알고리즘의 예1

- 정점 수 n = 5, 간선 수 m = 7

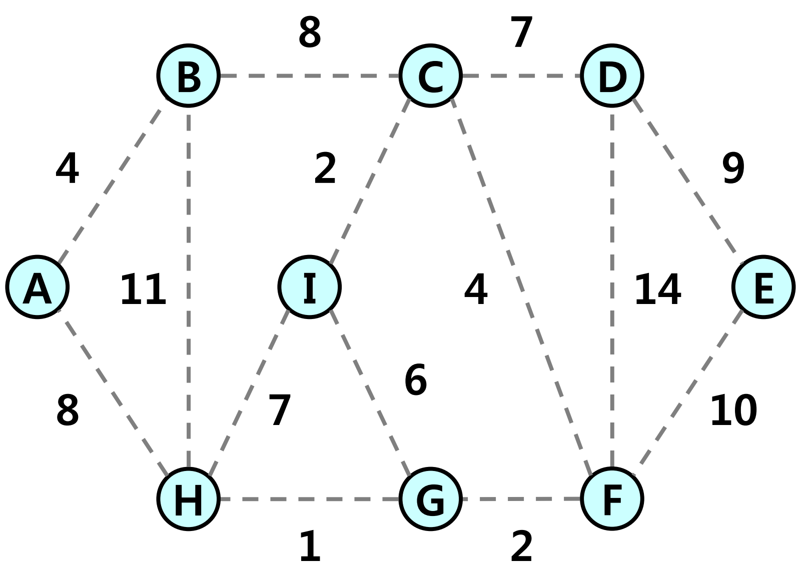


G1

edge	weight	사용
(D, E)	1	
(C, E)	3	
(C, D)	5	
(A, C)	7	
(A, B)	10	
(A, D)	10	
(B, D)	12	

Kruskal 알고리즘의 예2

- 정점 수 n = 9, 간선 수 m = 14



G2

edge	weight	사용
(G, H)	1	
(C, I)	2	
(F, G)	2	
(A, B)	4	
(C, F)	4	
(G, I)	6	
(C, D)	7	
(H, I)	7	
(A, H)	8	
(B, C)	8	
(D, E)	9	
(E, F)	10	
(B, H)	11	
(D, F)	14	

Prim 알고리즘

- Prim 알고리즘

그래프 G의 정렬되지 않은 간선에 대해

(1) 시작 정점을 선택하여 트리를 생성한다.

(2) 트리의 모든 정점에 부속된 모든 간선 중에서
가중치가 가장 작은 간선을 연결하여 트리를 확장한다.

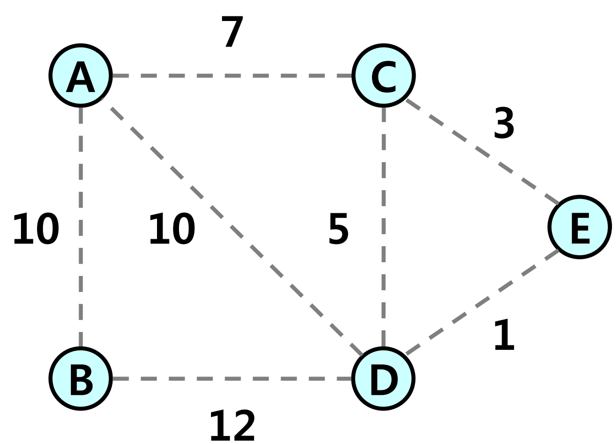
- 이때 사이클을 형성하는 간선은 삽입할 수 없으므로
그 다음으로 가중치가 작은 간선을 선택한다.

(3) 그래프 G에 n-1개의 간선을 삽입할 때까지 (2)를 반복한다.

(4) 그래프 G의 간선이 n-1개가 되면 최소 신장 트리가 완성된다.

Prim 알고리즘의 예1

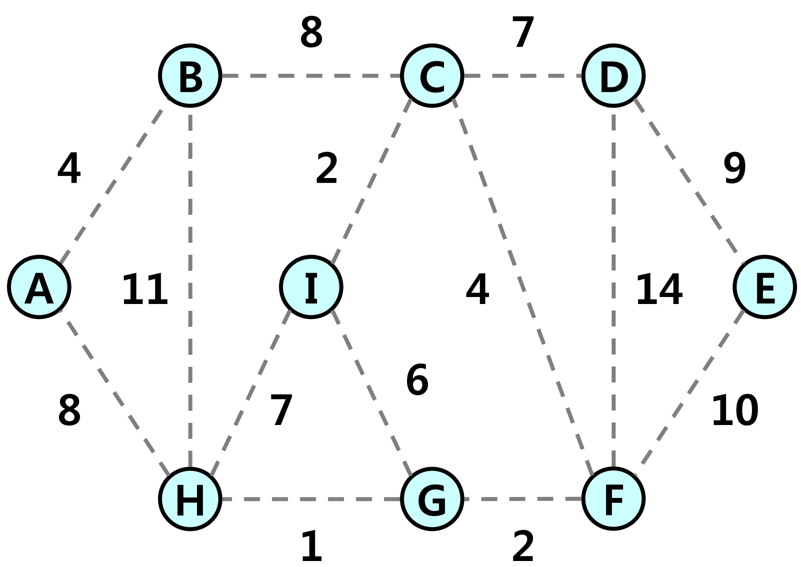
- 정점 수 n = 5, 간선 수 m = 7



G1

Prim 알고리즘의 예2

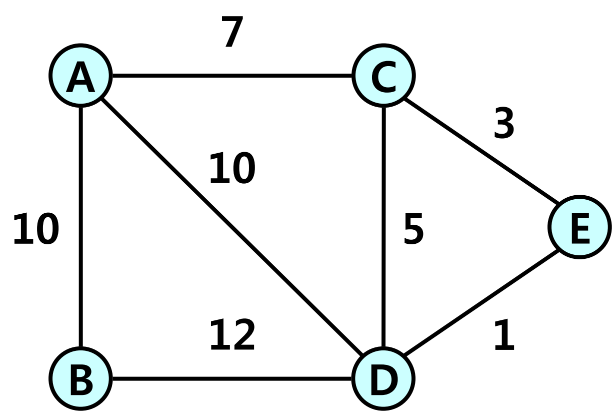
- 정점 수 n = 9, 간선 수 m = 14



G2

최소 신장 트리의 응용

- 도로망 건설 또는 네트워크 통신망 설계 등에 응용



도시간 통신망 건설 비용 그래프

Q&A