

9장 그래프



순서

9.1 그래프 추상 데이터 타입

9.2 그래프 표현

9.3 그래프 순회

그래프 추상 데이터 타입(1)

◆ 그래프의 정의

- $G = (V, E)$: 그래프 G 는 2개의 집합 V 와 E 로 구성
- V : 공백이 아닌 노드 또는 정점(vertex)의 유한집합
 - ◆ V 만 표현 : $V(G)$ 로 표기
- E : 상이한 두 정점을 잇는 간선(edge)의 유한집합
 - ◆ E 만 표현 : $E(G)$ 로 표기

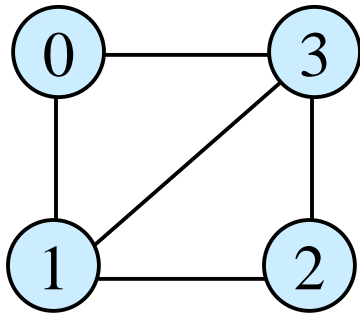
◆ 무방향 그래프(undirected graph)

- 간선을 표현하는 두 정점의 쌍에 순서가 없는 그래프
- $(v_0, v_1) = (v_1, v_0)$

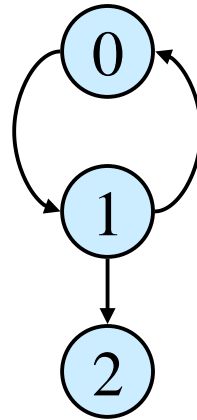
◆ 방향 그래프(directed graph)

- 유방향 그래프 또는 다이그래프(digraph)
- 간선을 표현하는 두 정점의 쌍에 순서가 있는 그래프
- $v_j \rightarrow v_k$ 를 $\langle v_j, v_k \rangle$ 로 표현 (v_j 는 꼬리(tail), v_k 는 머리(head))
- $\langle v_j, v_k \rangle \neq \langle v_k, v_j \rangle$

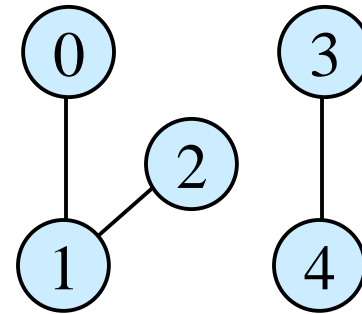
그래프 추상 데이터 타입(2)



G_1



G_2



G_3

$V(G_1)=\{0, 1, 2, 3\}$ $E(G_1)=\{(0, 1), (0, 3), (1, 2), (1, 3), (2, 3)\}$

$V(G_2)=\{0, 1, 2\}$ $E(G_2)=\{<0, 1>, <1, 0>, <1, 2>\}$

$V(G_3)=\{0, 1, 2, 3, 4\}$ $E(G_3)=\{(0, 1), (1, 2), (3, 4)\}$

그래프 예와 각 그래프의 정점 집합과 간선 집합

그래프 추상 데이터 타입(3)

◆ 단순 그래프(simple graph)

- 자기 루프(자신을 연결하는 루프)를 허용하지 않음
- 두 정점 사이에 최대 하나의 간선 존재
- 참고) 다중 그래프(multigraph) : 두 정점 사이에 복수 간선 가능

◆ 완전 그래프(complete graph)

- 최대 수의 간선을 가진 그래프
- 정점이 n 개일 때, 간선의 수는
무방향 그래프일 때 $n(n-1)/2$, 방향 그래프일 때 $n(n-1)$

그래프 추상 데이터 타입(4)

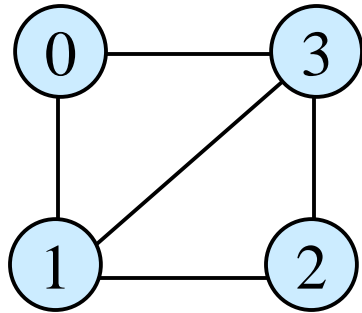
◆ 무방향 그래프의 한 간선 (v_j, v_k)

- v_j 와 v_k 는 서로 인접(adjacent)한다.
- 간선 (v_j, v_k)는 정점 v_j 와 v_k 에 부속(incident)한다.
- 예1) 그래프 G_1
 - ◆ 정점 1, 3 : 정점 0에 인접한다.
 - ◆ 간선 (0, 1), (1, 2), (1, 3) : 정점 1에 부속한다.
- 예2) 그래프 G_2
 - ◆ 아크 $\langle 0, 1 \rangle, \langle 1, 2 \rangle, \langle 1, 0 \rangle$: 정점 1에 부속한다.

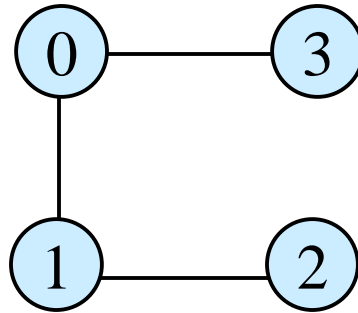
◆ 부분 그래프(subgraph)

- $V(G') \subseteq V(G)$ 이고, $E(G') \subseteq E(G)$ 인 그래프 G' 는 그래프 G 의 부분 그래프이다.

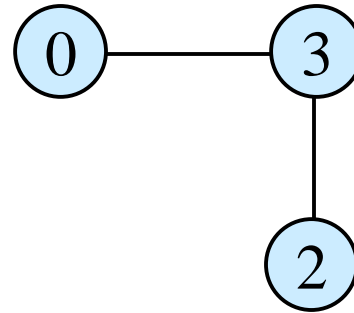
그래프 추상 데이터 타입(5)



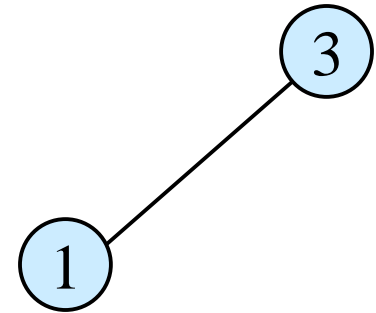
G_1



(i)

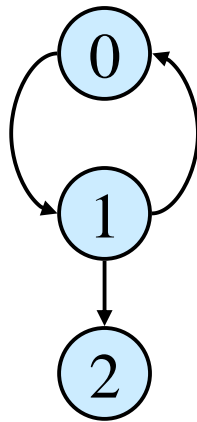


(ii)

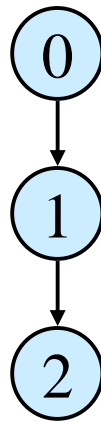


(iii)

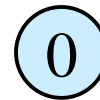
그래프 G_1 과 부분 그래프 예



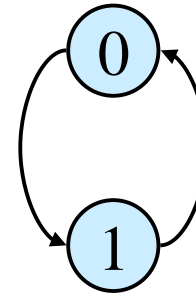
G_2



(i)



(ii)



(iii)

그래프 G_2 와 부분 그래프 예

그래프 추상 데이터 타입(6)

- ◆ 그래프 G 에서의 정점 u 로부터 v 까지의 경로(path)
 - 간선으로 연결된 정점의 순서 리스트 u, v_1, v_2, \dots, v, v
 - 여기서 $(u, v_1), (v_1, v_2), \dots, (v_k, v)$ 는 그래프 G 에 속한 간선
- ◆ 경로의 길이(path length)
 - 경로를 구성하는 간선의 수
- ◆ 단순 경로(simple path)
 - 모두 상이한 간선들로 구성된 경로
 - 예1) 그래프 G_1
 - ◆ 경로 0,1,3,2와 경로 0,1,3,1 : 길이는 모두 3
 - ◆ 첫번째 경로 : 단순 경로, 두 번째 경로 : 단순 경로 아님
 - 예2) 그래프 G_2
 - ◆ 0,1,2 : 단순 방향 경로, 0,1,2,1 : 경로가 아님

그래프 추상 데이터 타입(7)

◆ 사이클(cycle)

- 첫 번째 정점과 마지막 정점이 동일한 단순 경로
 - ◆ 무방향 그래프의 사이클 길이 : 3 이상
 - ◆ 방향 그래프의 사이클 길이 : 2 이상
- 예1) 무방향 그래프 G_1 의 경로 0, 1, 3, 0 : 사이클
- 예2) 방향 그래프 G_2 의 경로 0, 1, 0 : 사이클

◆ DAG(directed acyclic graph)

- 방향 그래프에서 사이클이 없는 그래프

그래프 추상 데이터 타입(8)

◆ 연결 그래프(**connected graph**)

- 서로 다른 모든 쌍의 정점들 사이에 경로가 있는 무방향 그래프
- 어느 한 정점에서부터 다른 어떤 정점으로의 경로 존재
- 트리 : 사이클이 없는 연결 그래프
- 예1) 그래프 G_1 : 연결 그래프
- 예2) 그래프 G_3 : 단절 그래프(**disconnected graph**)

◆ 연결 요소(**connected component**)

- 최대 연결 부분 그래프(**maximal connected subgraph**)
- 최대 : 최대의 정점과 최대의 간선
- 예) 그래프 G_3 : $\{0,1,2\}$ 와 $\{3,4\}$ 로 구성된 2개의 연결 요소

그래프 추상 데이터 타입(9)

◆ 강력 연결(strongly connected)

- 방향 그래프 G 에서 $V(G)$ 에 있는 서로 다른 모든 정점의 쌍 u 와 v 에 대해 u 에서 v 까지, 또한 v 에서 u 까지의 방향 경로가 존재

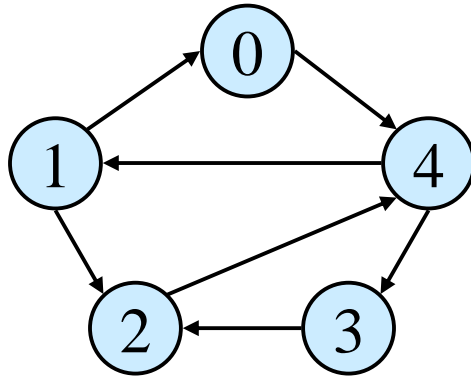
◆ 약한 연결(weakly connected)

- u 에서 v 까지, 또는 v 에서 u 까지 어느 하나의 경로만 존재

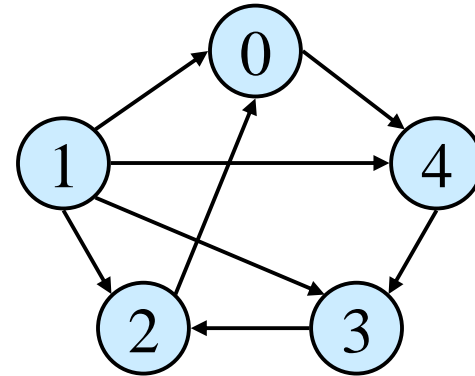
◆ 강력 연결 요소(strongly connected component)

- 강력 연결된 최대 부분 그래프

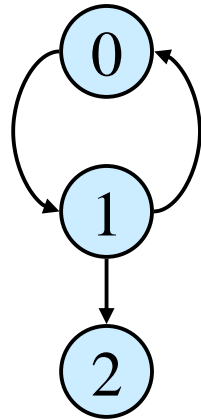
그래프 추상 데이터 타입(10)



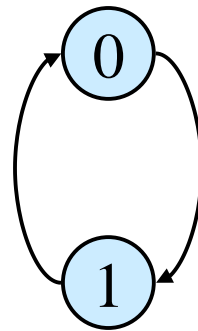
강력 연결 그래프 G_4



약한 연결 그래프 G_5



G_2



(i)



(ii)

G_2 의 두 강력 연결 요소

그래프 추상 데이터 타입(11)

◆ 무방향 그래프의 정점

- 차수(degree) : 무방향 그래프에서 그 정점에 부속된 간선의 수

◆ 방향 그래프의 정점 v

- 진입 차수(indegree) : 방향 그래프에서 정점 v 를 머리로 하는 간선의 수
- 진출 차수(outdegree) : 방향 그래프에서 정점 v 를 꼬리로 하는 간선의 수
- 선행자(predecessor) : 정점 v 를 머리로 하는 간선들의 정점(v_i)들, $\langle v_i, v \rangle$
- 후행자(successor) : 정점 v 를 꼬리로 하는 간선들의 정점(v_k)들 $\langle v, v_k \rangle$
- 예) 그래프 G_2
 - ◆ 정점 0 : 진입 차수 = 진출 차수 = 1
 - ◆ 정점 1 : 진입 차수 = 1, 진출 차수 = 2

그래프 추상 데이터 타입(12)

- ◆ n 개의 정점, e 개의 간선으로 된 그래프 G 에서 정점 i 의 차수가 $\text{degree}(i)$ 일 때,

$$e = (\sum_{i=0}^{n-1} \text{degree}(i)) / 2$$

- ◆ 그래프 G 에서 각 정점 u 에 대한 인접 집합(adjacency set), $\text{adjacency}(u)$

$$\text{adjacency}(u) = \{v \mid (u, v) \in E(G)\}$$

그래프 추상 데이터 타입(13)

◆ 그래프 추상 데이터 타입

ADT Graph

데이터 : 공백이 아닌 정점(vertex)의 유한집합(V)과 간선(edge)의 집합(E).
여기서 간선은 두 정점의 쌍.

연산 내용 : $G \in \text{Graph}$, $u, v \in V$;

$\text{createG}() ::= \text{create an empty graph};$

$\text{insertVertex}(G, v) ::= \text{insert vertex } v \text{ into } G;$

$\text{insertEdge}(G, u, v) ::= \text{insert edge } (u, v) \text{ into } G;$

$\text{deleteVertex}(G, v) ::= \text{delete vertex } v \text{ and all edges incident on } v \text{ from } G;$

$\text{deleteEdge}(G, u, v) ::= \text{delete edge } (u, v) \text{ from } G;$

$\text{isEmpty}(G) ::= \text{if } G \text{ has no vertex then return true, else return false};$

$\text{adjacent}(G, v) ::= \text{return set of all vertices adjacent to } v;$

End Graph

그래프 표현

◆ 그래프 표현 방법

- 인접 행렬(adjacency matrix)
- 인접 리스트(adjacency list)
- 인접 다중 리스트(adjacency multilist)

◆ 그래프에 수행시키려는 연산과 적용하려는 응용에 따라 선택

인접 행렬(1)

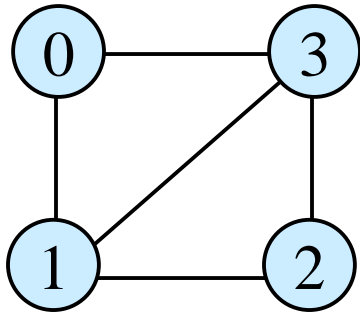
◆ 인접 행렬(adjacency matrix)

- $n \geq 1$ 개의 정점을 가지는 그래프 $G = (V, E)$ 에 대해, 크기가 $n \times n$ 인 2차원 배열 $a[n, n]$

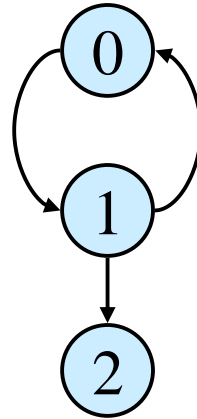
$$a[i, j] = \begin{cases} 1, & (i, j) \in E(G) \\ 0, & (i, j) \notin E(G) \end{cases}$$

- 부속 행렬(incidence matrix)이라고도 함
- 인접 행렬로 표현하는데 필요한 공간 : n^2 비트
 - ◆ 무방향 그래프 : 행렬의 상위 삼각이나 하위 삼각만 저장한다면 거의 반 정도의 공간을 절약
- 인접 행렬의 정보
 - ◆ 무방향 그래프 : 행 i 의 합은 정점 i 의 차수
 - ◆ 방향 그래프 : 행 i 의 합은 정점 i 의 진출 차수, 열 i 의 합은 정점 i 의 진입 차수

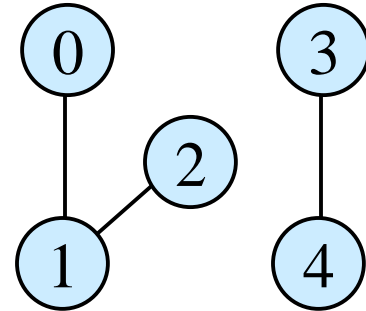
인접 행렬(2)



G_1



G_2



G_3

	[0]	[1]	[2]	[3]
[0]	0	1	0	1
[1]	1	0	1	1
[2]	0	1	0	1
[3]	1	1	1	0

$a[4, 4]$

	[0]	[1]	[2]
[0]	0	1	0
[1]	1	0	1
[2]	0	0	0

$a[3, 3]$

	[0]	[1]	[2]	[3]	[4]
[0]	0	1	0	0	0
[1]	1	0	1	0	0
[2]	0	1	0	0	0
[3]	0	0	0	0	1
[4]	0	0	0	1	0

$a[5, 5]$

그래프 G_1, G_2, G_3 에 대한 인접 행렬 표현

인접 리스트(1)

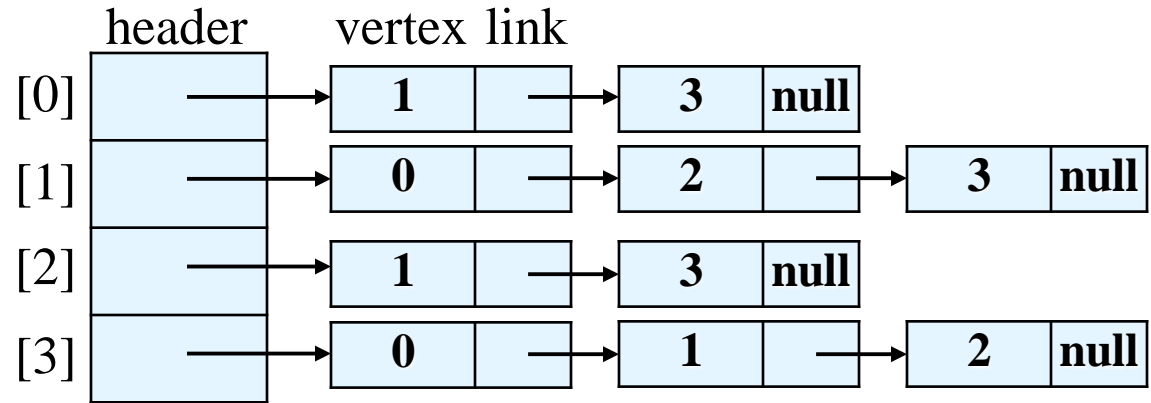
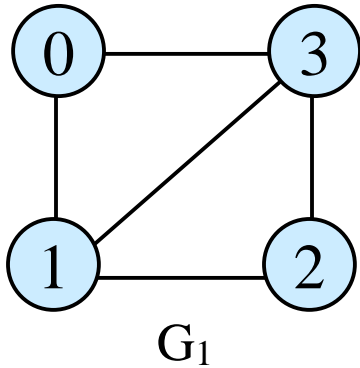
◆ 인접 리스트(adjacency list)

- n 개의 정점 각각에 대한 인접한 정점들을 리스트로 만듦
- 인접 리스트의 구현
 - ◆ 연결 리스트
 - ◆ 순차 표현

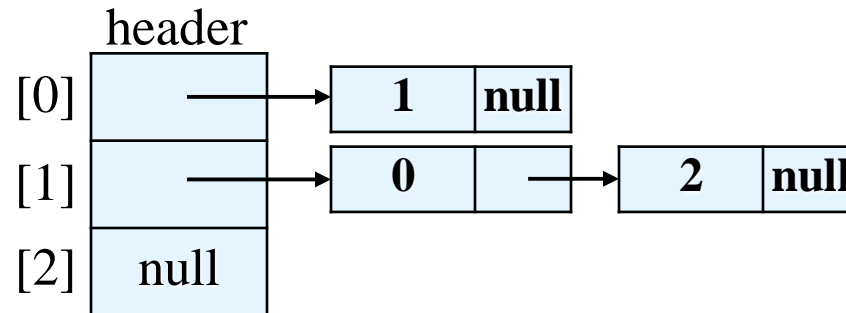
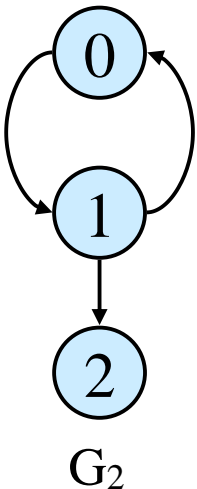
◆ 연결 리스트로 구현한 인접 리스트

- 각각의 정점의 리스트 : 헤더 노드와 vertex 필드, link 필드로 구성된 리스트 노드로 이루어짐
- n 개의 정점과 e 개의 간선을 가진 그래프에서
 - ◆ 무방향 그래프는 n 개의 헤더 노드와 $2e$ 개의 리스트 노드 필요
 - ◆ 방향 그래프는 n 개의 헤더 노드와 e 개의 리스트 노드 필요

인접 리스트(2)



(a) G_1 에 대한 인접 리스트



(b) G_2 에 대한 인접 리스트

인접 리스트(3)

◆ 순차 표현으로 구현한 인접 리스트

- 포인터가 아닌 리스트의 노드를 순차적으로 묶어 배열로 저장
- n 개의 정점과 e 개의 간선을 가진 그래프를 $\text{vertex}[n+2e+1]$ 로 표현
- $\text{vertex}[i]$: 정점 i 에 대한 인접 리스트의 시작점
- $\text{vertex}[n]$: 배열의 크기인 $n+2e+1$ 를 저장
- 정점 i 에 인접한 정점들은 $\text{vertex}[\text{vertex}[i]], \dots, \text{vertex}[\text{vertex}[i+1]-1], 0 \leq i < n$ 에 저장

인접 리스트(4)

◆ 그래프 G1에 대한 순차 표현

vertex	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	[12]	[13]	[14]
	5	7	10	12	15	1	3	0	2	3	1	3	0	1	2

리스트 시작과 배열 크기 정보 간선 정보

- 배열 vertex[] 크기 : 15 (::n=4, e=5), vertex[4]에 저장
- 정점 0에 대한 인접 리스트 : vertex[5]부터 저장됨
 - ◆ vertex[0] : 인덱스 5를 저장
 - ◆ vertex[5], vertex[6] : 두 개의 인접 정점 1과 3을 저장
- 정점 1에 대한 인접 리스트 : vertex[7]부터 저장됨
 - ◆ vertex[1] : 인덱스 7을 저장
 - ◆ Vertex[7], vertex[8], vertex[9] : 인접 정점 0, 2, 3을 저장
- 위의 방법으로 정점 3에 대한 인접 리스트까지 모두 저장

인접 리스트(5)

◆ 정점의 차수

- 정점에 소속된 간선의 수
- 정점의 인접 리스트에 있는 노드 수를 계산
- 전체 간선 수 계산에 걸리는 시간 : $O(n+e)$

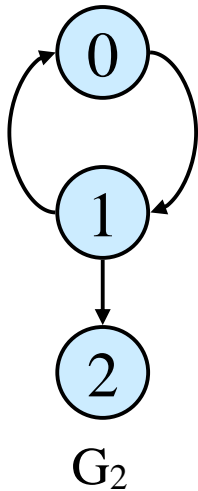
◆ 방향 그래프에서의 각 정점의 차수




- 진출 차수 : 단순히 인접 리스트의 노드 수만 계산
- 인접 리스트로부터 진입 차수 : 계산이 복잡하므로, 역인접 리스트를 별도로 유지

◆ 역인접 리스트(**inverse adjacency list**)

- 각 정점 i 에 대해, 정점 i 로 진입하는 모든 정점 각각에 대한 노드를 포함시킨 리스트

인접 리스트(6)



	header		
[0]		1	null
[1]		0	null
[2]		1	null

그래프 G_2 에 대한 역인접 리스트

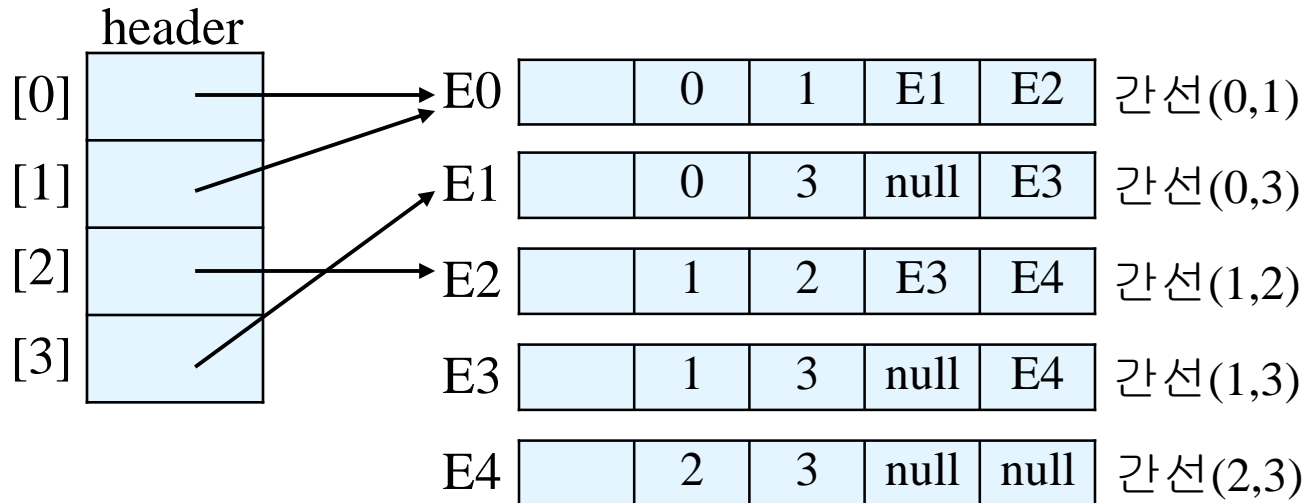
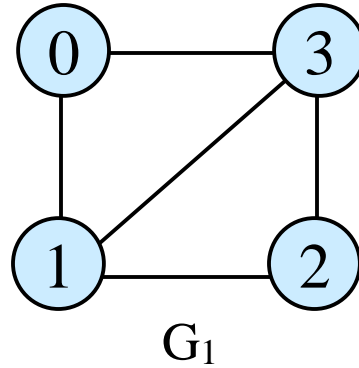
인접 다중 리스트(1)

◆ 인접 다중 리스트(adjacency multilist)

- 다중 리스트
 - ◆ 노드들을 여러 리스트들이 공유하는 리스트
 - ◆ 특정 간선에 대한 접근 여부를 표시하는 데 편리
- 간선에 부속된 두 정점 각각에 대한 인접 리스트를 다중 리스트로 유지하여, 하나의 간선을 두 개의 리스트가 공유하는 리스트
- 간선 (i, j) 를 표현하는 노드 구조
 - ◆ M : 간선이 이미 검사되었는지 여부를 표시하는 마크 비트
 - ◆ i-link, j-link : 각각 정점 i, j 에 대한 인접 리스트의 링크

M	i	j	i-Link	j-Link
---	---	---	--------	--------

인접 다중 리스트(2)



G_1 에 대한 인접 다중 리스트

인접 다중 리스트(3)

◆ G1에 대한 인접 다중 리스트의 식별

- 정점 0의 경우
 - ◆ 정점 0의 헤더로부터 노드 E0를 따라감
 - ◆ 노드 E0에서 정점 0을 포함한 필드 = $i \rightarrow i\text{-link}$ 를 따라 E1으로 감
 - ◆ 노드 E1의 첫 번째 i 필드가 정점 0 포함 \rightarrow 다시 $i\text{-link}$ 를 따라감
 - ◆ 이 때 링크값이 널 \rightarrow 여기서 리스트가 끝남
- 위의 방법으로 모든 정점들을 식별

정점 0 : E0 \rightarrow E1

정점 1 : E0 \rightarrow E2 \rightarrow E4

정점 2 : E2 \rightarrow E4

정점 3 : E1 \rightarrow E3 \rightarrow E4

그래프 순회

◆ 그래프 순회

- 주어진 어떤 정점을 출발하여 체계적으로 그래프의 모든 정점들을 순회하는 것

◆ 그래프 순회의 종류

- 깊이 우선 탐색(DFS)
- 너비 우선 탐색(BFS)

◆ 그래프 순회의 응용

- 연결 요소
- 신장 트리

깊이 우선 탐색(1)

◆ 깊이 우선 탐색(depth first search : DFS) 수행

- (1) 정점 i 를 방문한다.
- (2) 정점 i 에 인접한 정점 중에서 아직 방문하지 않은 정점이 있으면, 이 정점들을 모두 스택에 저장한다.
- (3) 스택에서 정점을 삭제하여 새로운 i 를 설정하고, 단계 (1)을 수행한다.
- (4) 스택이 공백이 되면 연산을 종료한다.

◆ 정점 방문 여부를 표시

- 배열 $visited[n]$ 을 이용하여 표현

$$visited[i] = \begin{cases} \text{true,} & \text{방문하였음} \\ \text{false,} & \text{방문하지 않았음} \end{cases}$$

깊이 우선 탐색(2)

◆ 깊이 우선 탐색 알고리즘

DFS(i)

// i 는 시작 정점

for ($j \leftarrow 0$; $j < n$; $j \leftarrow j + 1$) **do** {

$visited[j] \leftarrow false$; // 모든 정점을 방문 안한 것으로 마크

}

createStack(); //방문할 정점을 저장하는 스택

push(Stack, i); // 시작 정점 i 를 스택에 저장

while (**not** isEmpty(Stack)) **do** { // 스택이 공백이 될 때까지 반복 처리

$j \leftarrow pop(Stack)$;

if ($visited[j] = false$) **then** { //정점 j를 아직 방문하지 않았다면

 visit j; // 직접 j를 방문하고

$visited[j] \leftarrow true$; // 방문 한 것으로 마크

for (each $k \in adjacency(j)$) **do** { // 정점 j에 인접한 정점 중에서

if ($visited[k] = false$) **then** // 아직 방문하지 않은 정점들을

 push(Stack, k); // 스택에 저장

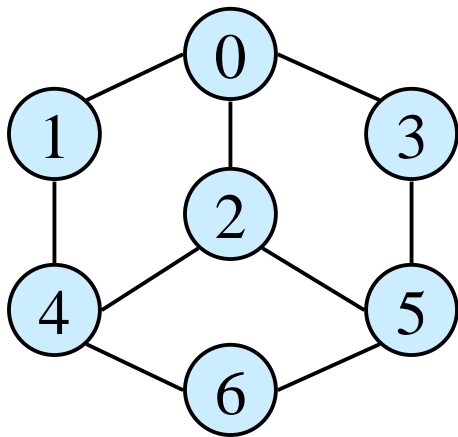
 }

 }

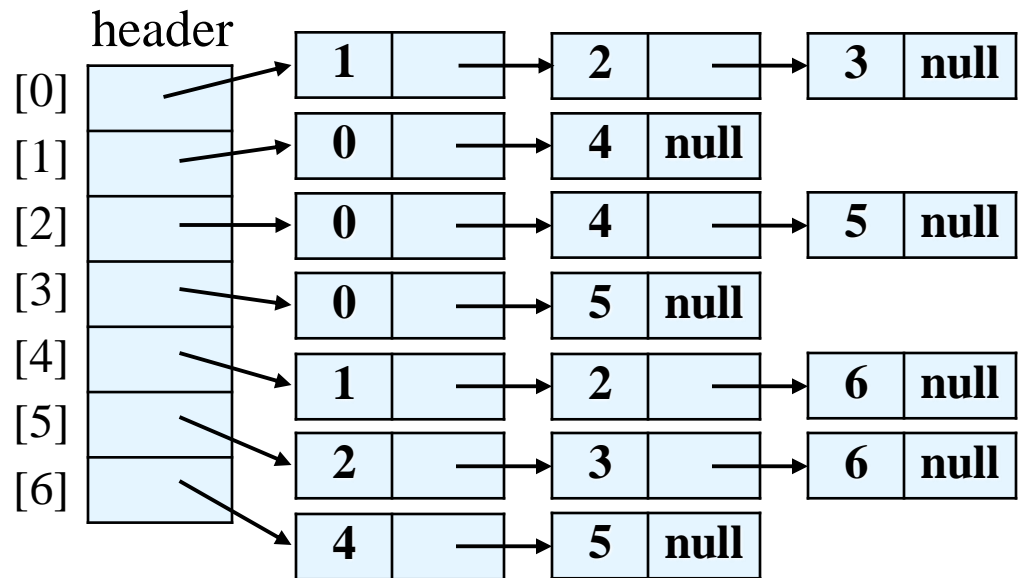
}

end DFS()

깊이 우선 탐색(3)

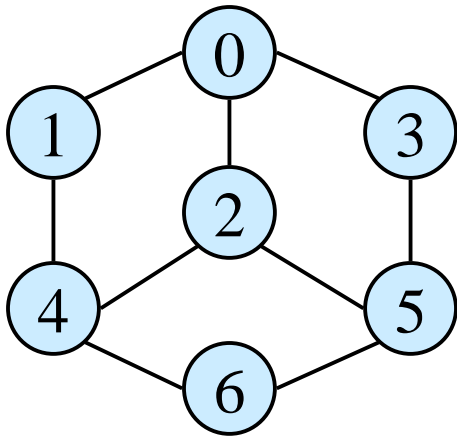


탐색을 위한 그래프 G

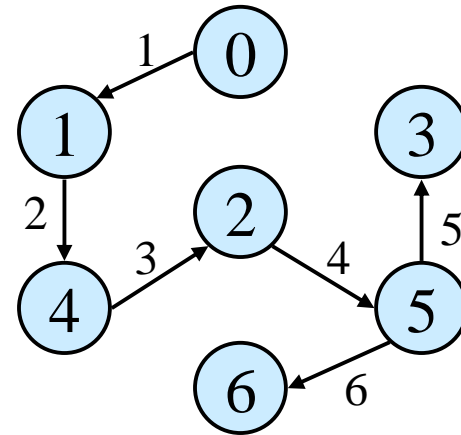


그래프 G에 대한 인접 리스트 표현

깊이 우선 탐색(4)



탐색을 위한 그래프 G



그래프 G에 대한
깊이 우선 탐색 경로

너비 우선 탐색(1)

◆ 너비 우선 탐색(breadth first search ; BFS) 수행

- (1) 정점 i 를 방문한다.
- (2) 정점 i 에 인접한 정점 중에서 아직 방문하지 않은 정점이 있으면, 이 정점들을 모두 큐에 저장한다.
- (3) 큐에서 정점을 삭제하여 새로운 i 를 설정하고, 단계 (1)을 수행한다.
- (4) 큐가 공백이 되면 연산을 종료한다.

◆ 정점 방문 여부를 표시

- 깊이 우선 탐색과 마찬가지로 배열 `visited[n]`을 이용

너비 우선 탐색(2)

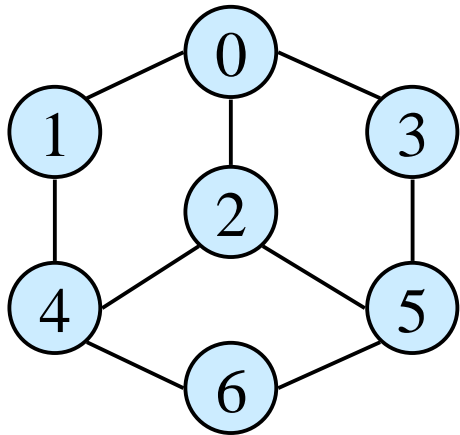
◆ 너비 우선 탐색 알고리즘

BFS(i)

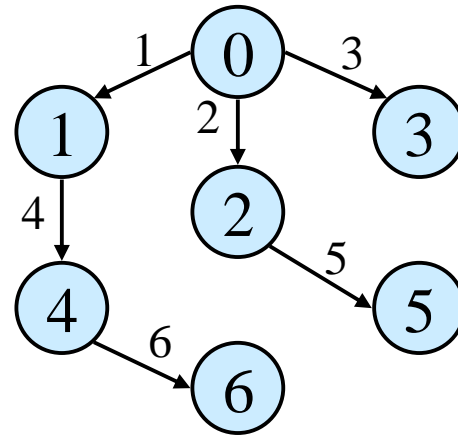
// i는 시작 정점

```
for (j ← 0; j < n; j ← j + 1) do {  
    visited[j] ← false; // 모든 정점을 방문 안 한 것으로 마크  
}  
createQ(); // 방문할 정점을 저장하는 큐  
enqueue(Q, i);  
while (not isEmpty(Q)) do {  
    j ← dequeue(Q);  
    if (visited[j] = false) then {  
        visit j;  
        visited[j] ← true;  
    }  
    for (each k ∈ adjacency(j)) do {  
        if (visited[k] = false) then {  
            enqueue(Q, k);  
        }  
    }  
}  
end BFS()
```

너비 우선 탐색(3)



탐색을 위한 그래프 G



그래프 G에 대한
너비 우선 탐색 경로

연결 요소(1)

◆ 연결 그래프 여부 판별

- DFS나 BFS 알고리즘 이용
- 무방향 그래프 G 에서 하나의 정점 i 에서 시작하여 DFS(or BFS)로 방문한 노드집합 $V(\text{DFS}(G, i))$ 가 $V(G)$ 와 같으면 G 는 연결 그래프.

$V(\text{DFS}(G, i)) = V(G)$: 연결 그래프, 하나의 연결 요소

$V(\text{DFS}(G, i)) \subset V(G)$: 단절 그래프, 둘 이상의 연결 요소

◆ 연결 요소 찾기

- 정점 i 에 대해 DFS (or BFS) 수행
- 둘 이상의 연결 요소가 있는 경우, 나머지 정점 j 에 대해 DFS(or BFS) 반복 수행

연결 요소(2)

◆ 연결 요소를 찾는 알고리즘

```
dfsComponent(G, n)  // G=(V,E), n은 G의 정점 수
  for (i ← 0; i < n; i ← i + 1) do {
    visited[i] ← false;
  }
  for (i ← 0; i < n; i ← i + 1) do {
    // 모든 정점 0, 1, ..., n-1에 대해 연결 요소 검사
    if (visited[i] = false) then {
      print("new component");
      DFS(i);  // 정점 i가 포함된 연결 요소를 탐색
    }
  }
end dfsComponent()
```

- DFS(i)를 BFS(i)로 대체해도 무방

신장 트리(1)

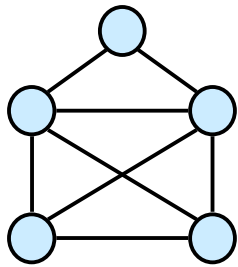
◆ 트리 간선(tree edge)

- G가 연결 그래프일 때
 - ◆ DFS나 BFS는 G의 모든 정점 방문
 - ◆ G의 간선 : 방문에 사용한 간선들과 그렇지 않은 간선들로 나뉨
- 방문에 사용된 간선 (j, k) 의 집합을 T라 할 때
 - ◆ DFS와 BFS 알고리즘의 for속의 if-then 절에 명령문 $T \leftarrow T \cup \{(j, k)\}$ 삽입시켜 구할 수 있음
 - ◆ T에 있는 간선들을 전부 결합시키면 그래프 G의 모든 정점들을 포함한 트리가 됨
- 이러한 간선을 트리 간선이라 함

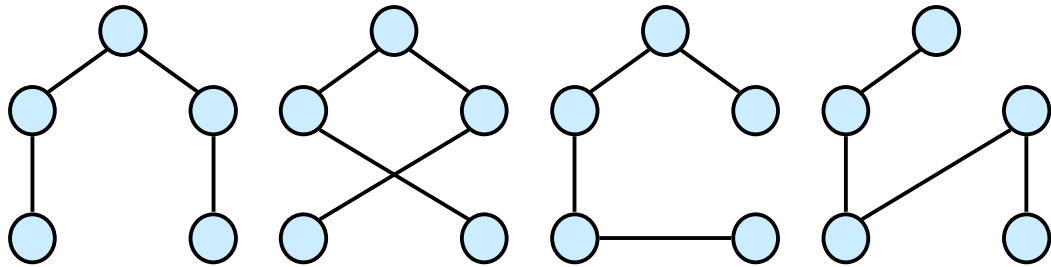
신장 트리(2)

◆ 신장 트리(spanning tree)

- 그래프 G 에서 $E(G)$ 에 있는 간선과 $V(G)$ 에 있는 모든 정점들로 구성된 트리
- DFS, BFS에 사용된 간선 집합 T 는 그래프 G 의 신장 트리를 의미
- 주어진 그래프 G 에 대한 신장 트리는 유일하지 않음



(a) 연결 그래프 G

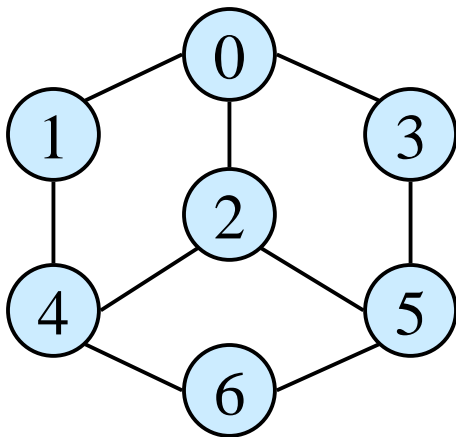


(b) 신장 트리

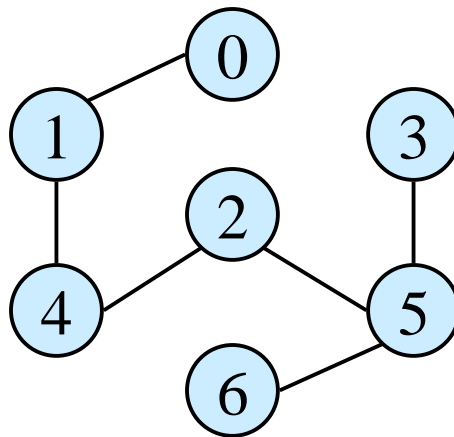
신장 트리(3)

◆ 신장 트리의 종류

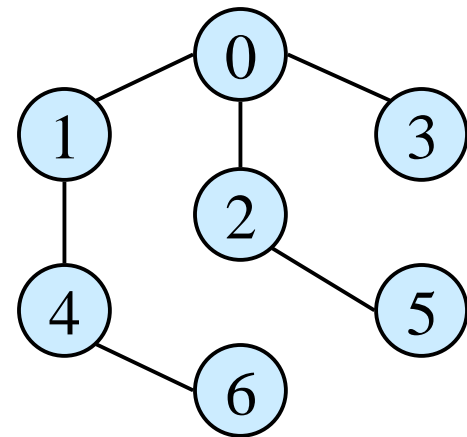
- 깊이 우선 신장 트리(depth first spanning tree) : DFS 사용
- 너비 우선 신장 트리(breadth first spanning tree) : BFS 사용



(a) 연결 그래프 G



(b) DFS(0) 신장 트리



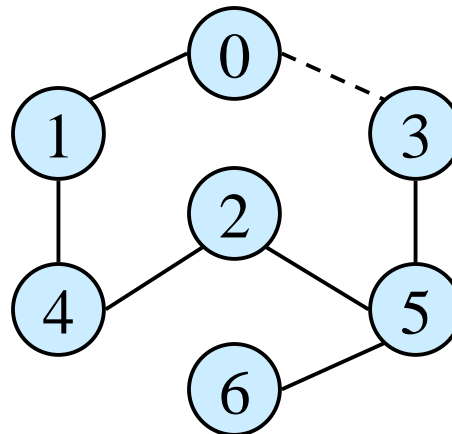
(c) BFS(0) 신장 트리

연결 그래프 G와 신장 트리

신장 트리(4)

◆ 비트리 간선(nontree edge) 집합(NT)

- 신장 트리에 사용되지 않은 간선들의 집합
- NT에 있는 임의의 간선 (i, j) 를 신장 트리 T 에 첨가시키면 사이클이 만들어져 더 이상 트리가 아님
- 예) DFS(0) 신장 트리
 - ◆ 간선 $(0, 3)$ 첨가 : 0, 1, 4, 2, 5, 3, 0으로 구성된 사이클 형성



신장 트리(5)

◆ 최소 연결 부분 그래프(minimal connected subgraph)

- G 의 부분 그래프 G' 중 다음 조건을 만족하는 그래프

- (1) $V(G') = V(G)$
- (2) $E(G') \subseteq E(G)$
- (3) G' 는 연결 그래프
- (4) G' 는 최소의 간선 수를 포함

- 신장 트리는 최소 연결 부분 그래프로서, $n-1$ 개의 간선을 가짐
 - ◆ n 개의 정점을 가진 그래프는 최소한 $n-1$ 개의 간선 필요
 - ◆ $n-1$ 개의 간선을 가진 연결 그래프는 트리
- 응용 : 통신 네트워크 설계
 - ◆ 도시간 네트워크 설계에서 최소 링크 수를 구하는 데 사용