9주차 1차시 그래프의 정의

【학습목표】

1. 그래프의 연결성을 이해할 수 있다.
2. 방향그래프의 구현을 이해할 수 있다.

학습내용1 : 그래프의 표현

1. 그래프의 정의

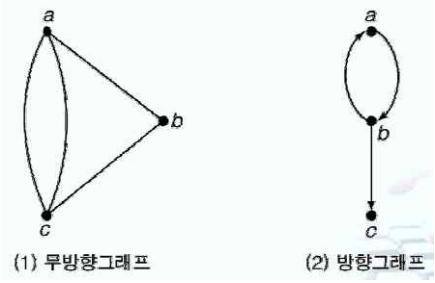
그래프는 “꼭지점의 집합 V와 변의 집합 E의 순서쌍(V, E)로 정의, 변은 두 곡지점을 잇는다” , 예를 들어 두 꼭지점 V1과 V2를 잇는 (V1, V2)로 표현할 수 있다.

① 그래프의 표현

★ 그래프 표현
◎ 그래프 G = (V, E)
◎ V: 정점의 집합(원으로 표시 된 것.)
◎ E: 간선의 집합(정점과 정점을 연결한 선.)
◎ 2차원 배열을 이용하여 인접 행렬이나 연결리스트 형태의 인접 리스트로 표현 가능하다

이러한 그래프는 그 형태에 따라 여러 가지 종류로 나뉘다. 크게는 무 방향그래프((Undirected Graph)와 방향 그래프(Directed Graph)로 나뉜다.
변이 방향성을 띠지 않는 그래프를 무 방향 그래프라고 하며, 방향성을 띠는 그래프를 방향그래프라 한다.

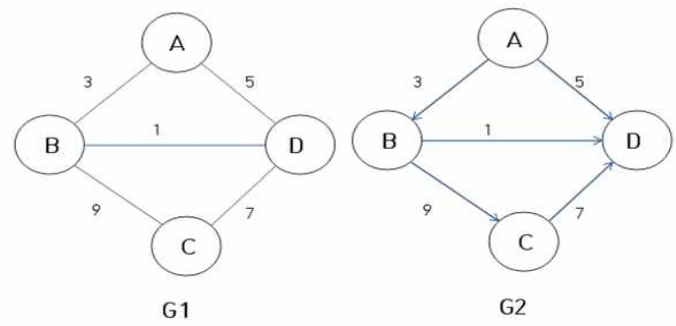
다음그림의 그래프(1), (2)를 집합으로 나타내시오



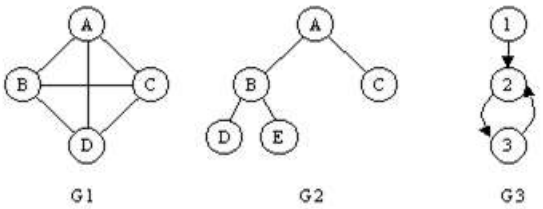
- 그래프(1)
주어진 그래프를 G라고 하자. 그래프 G는 무방향 그래프이다.
G = (V, E)
V = {a, b, c}
E = { (a,b),(b,c),(a,c),(a,c)}

- 그래프(2)
주어진 그래프를 H라고 하자. 그래프 H는 방향 그래프이므로 순서에 유의하여야 한다
H =<V,E>
V = {a,b,c}
E = {<a,b>,<b,a>,<b,c>}.

가중 그래프(Weighted Graph)는 정점(Vertex)를 연결하는 간선(Edge)에 가중치를 부여한 것입니다.(G2 가중방향그래프(네트워크))



[예제]
다음그림의 그래프(G1), (G2), (G3)를 집합으로 나타내시오

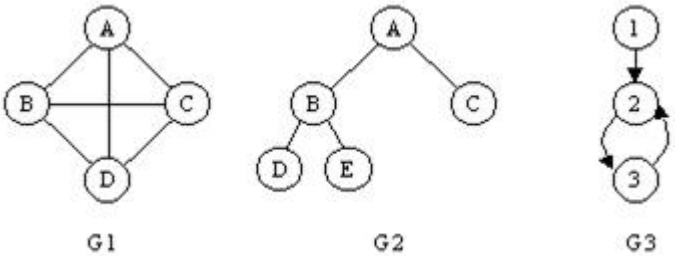


[해설]
- G1과 G2는 무방향 그래프이고, G3은 방향 그래프이다.
- 그래프 G1, G2, G3의 정점들의 집합 V(G)와 간선들의 집합 E(G)

V(G1) = {A, B, C, D}
E(G1) = {(A, B), (A, C), (A, D), (B, C), (B, D), (C, D)}
V(G2) = {A, B, C, D, E}
E(G2) = {(A, B), (A, C), (B, D), (B, E)}
V(G3) = {1, 2, 3}
E(G3) = {<1, 2>, <2, 3>, <3, 2>}

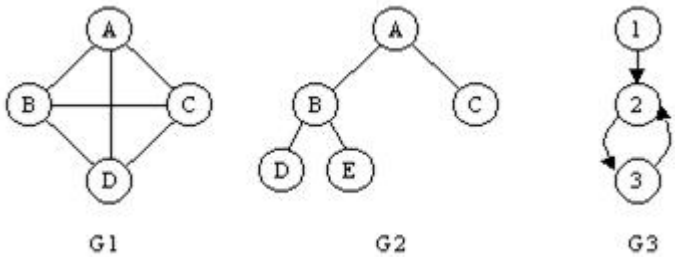
2. 그래프의 종류

① 인접(adjacent)
- 간선(v, w)가 E(G)에 속한 간선 → 정점 v와 정점 w는 인접
- 아래그림 G1에서 정점 A에 인접된 정점 : B, C, D



그래프의 예

② 부속(incident)
- v, w가 인접되어 있을 때, 간선(v, w) : 정점 v, w에 부속된 것
- 아래그림 G2에서 정점 B에 부속된 간선들 : (B, A), (B, D), (B, E)
- 그래프의 예

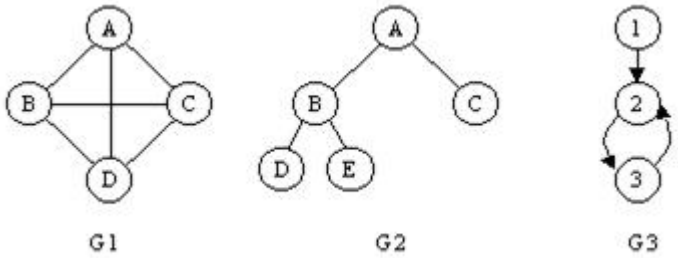


③ 단순 경로(simple path)

• 경로 상에 있는 정점들 중에서 첫 번째 정점과 마지막 정점을 제외한 모든 정점들이 서로 다를 때

• 아래그림 G1에서 ABD, ABDA : 단순 경로, ABDBA : 단순 경로가 아님

• 그래프의 예

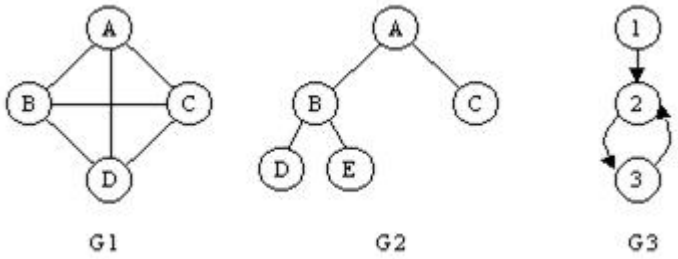


④ 사이클(cycle)

• 사이클 : 첫 번째 정점과 마지막 정점이 같은 단순 경로

• 아래그림 G1에서 ABDA : 단순 경로이고 사이클임, ABDCBA : 단순 경로는 아니지만 사이클임

• 그래프의 예



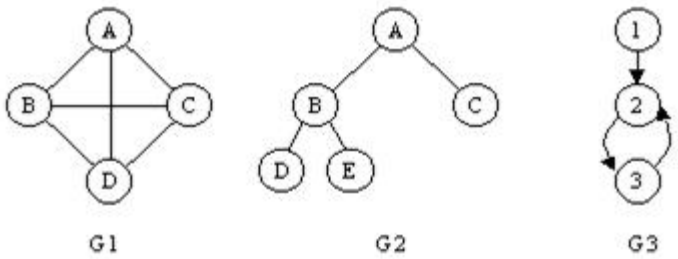
⑤ 연결(connected)

• 무방한 그래프에서 두 정점 v에서 w로 가는 경로가 존재

- 정점 v와 정점 w는 연결되었다고 함

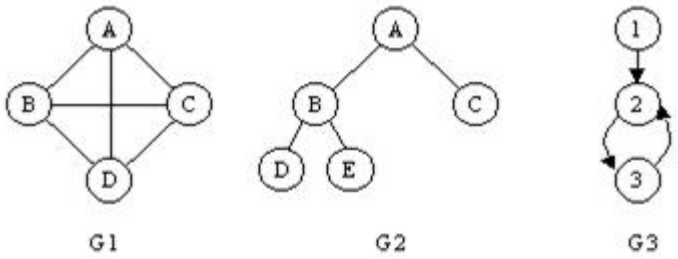
• 아래그림 G1에서 A, B, C, D 모든 정점은 서로 연결되어 있음

• 그래프의 예



⑥ 차수(degree)

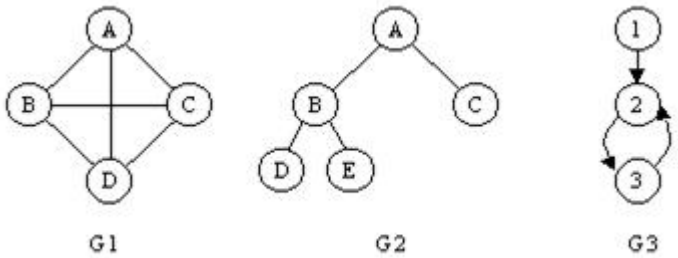
- 임의의 정점에 부속된 간선들의 수
- 아래그림 G2에서 정점 B의 차수는 2임
- 그래프의 예



G1 G2 G3

⑦ 진입 차수(indegree)

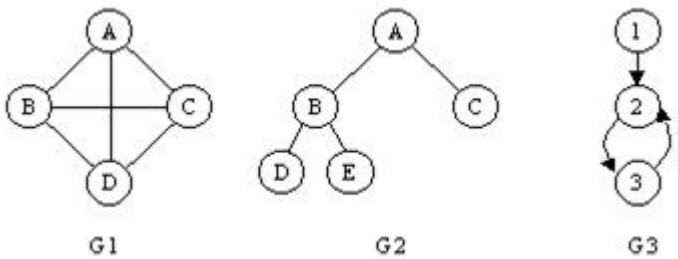
- 방향 그래프 G에서 임의의 정점 v가 머리(head)가 되는 간선들의 개수
- 아래그림 G3에서 정점 2의 진입 차수 : 2
- 그래프의 예



G1 G2 G3

⑧ 진출 차수(outdegree)

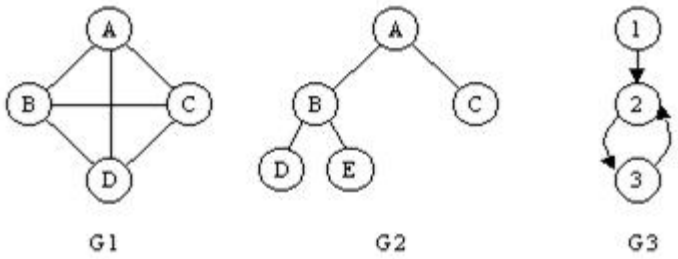
- 방향 그래프 G에서 임의의 정점 v가 꼬리가 되는 간선의 개수
- 아래그림 G3에서 정점 2의 진출 차수 : 1
- 그래프의 예



G1 G2 G3

⑨ 연결 그래프(connected graph)

- 그래프 G에서 그래프에 속한 모든 정점이 연결되어 있어서 Vi, Vj의 모든 쌍에 대하여 경로가 존재하는 그래프
- 아래그림 G1, G2, G3 그래프는 모두 연결 그래프임
- 그래프의 예

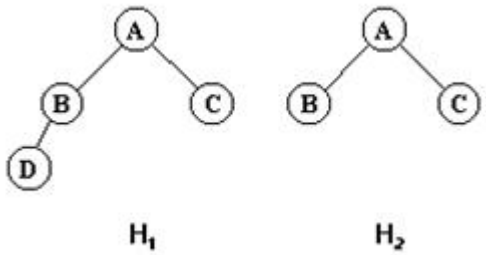


⑩ 단절 그래프(disconnected graph)

- 단절 그래프 : 연결 그래프가 아닌 그래프
- 아래그림에서 그래프 G4에서 전체가 하나의 그래프임

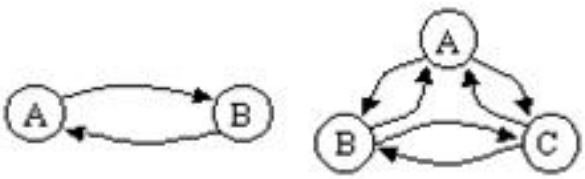
H1과 H2로 단절이 되어 있으므로 → 그래프 G4 : 단절 그래프

- 단절 그래프



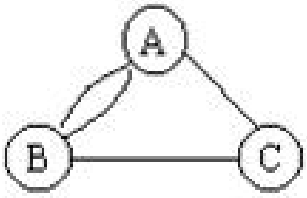
⑪ 강력 연결 그래프(strongly connected graph)

- 방향 그래프 G에서 V(G)에 속한 서로 다른 두 정점 vi, vj의 모든 쌍에 대해 <vi, vj>와 <vj, vi>인 방향 경로가 존재하는 방향 그래프
- 모두 강력 연결 그래프



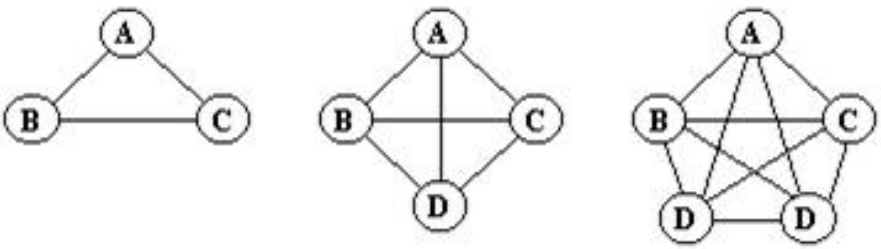
⑫ 다중 그래프

- 그래프 G의 임의의 두 정점 사이에 두 개 이상의 간선이 존재하는 그래프
- 아래그림 A와 B 사이에 2개의 간선이 있으므로 다중 그래프
- 다중 그래프



⑬ 완전 그래프(complete graph)

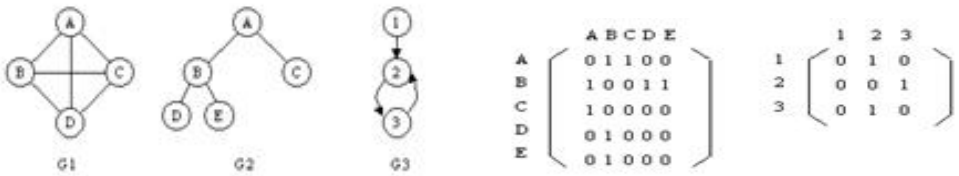
- 그래프에서 모든 정점과 정점 사이에 간선이 존재하여 임의의 정점에서 다른 모든 정점에 간선이 존재하는 그래프
- 완전 연결 그래프(fully connected graph)
- 무방한 그래프 G의 정점의 개수가 n개 일 때 간의 최대수 : n(n-1)/2개
- 방향 그래프 G에서는 간선의 수가 n(n-1)개인 그래프
- 완전 그래프



3. 인접 행렬에 의한 방법

인접행렬(Adjacency Matrix)은 정점 집합 V(G)={ V1, V2… Vn}인 그래프 G=(V(G),E(G))의 인접 행렬(adjacency matrix)은 그래프를 구성하는 각 정점들 간의 인접 여부를 n×n의 2차원 배열로 표현한 것이다.
2차원 배열 A(i, j)에서 연결선 (Vi, Vj)가 E(G)에 속하면 A(i, j)=1이 되고, E(G)에 속하지 않으면 A(i, j)=0이 된다.
아래그림에서 그래프 G2와 G3 인접 행렬로 표현한 예이다.

* 인접 행렬을 이용한 그래프의 표현

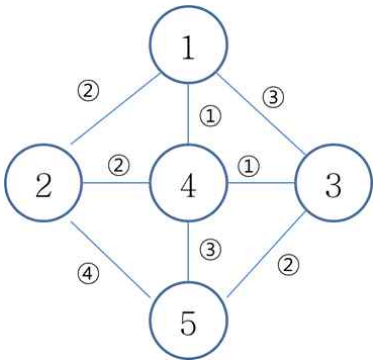


4. 인접 행렬

n개의 원소로 구성된 정점들의 집합을 V={1,2,3,,,n}이라 하자. 이때 인접 행렬 A=(n*n)의 임의의 원소 A[i][j]의 값이 참이면 정점 i와 j사이에는 간선이 존재함을 의미하고 A[i][j]의 값이 거짓이면정점 i와 j사이에는 간선이 없다는 것을 의미. 경우에 따라서 참을 1로 표현하고, 거짓을 0으로 표현할 수도 있음. i와 j가 동일한 대각선상의 원소 A[i][j]의 값은 0으로 정함

① 인접행렬 가중그래프(weighted graph)
간선 상에 값이 주어진 그래프

② 가중 그래프의 인접 행렬 표현법 예제



	①	②	③	④	⑤
①	**0**	**2**	**3**	**1**	**∞**
②	2	0	∞	2	4
③	3	∞	0	1	2
④	1	2	1	0	3
⑤	∞	4	2	3	0

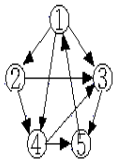
5. 인접 리스트(Adjacency List)

인접 리스트는 연결 리스트(linked list)를 사용하여 그래프의 구조를 표현하는 방법으로 그래프의 각 정점에 인접한 정점들을 각각 한 노드에 보관하여 한 개의 연결 리스트를 구성하여 각 리스트에 대한 포인터를 1차원 배열로 저장한 구조이다.

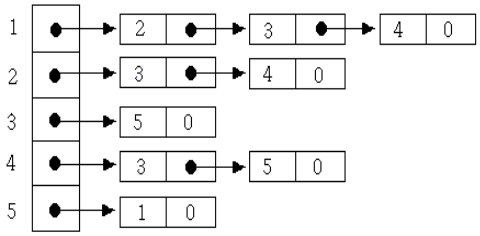
인접 리스트 표현법
- 그래프 포인터를 사용하여 인접리스트로 표현. 임의의 정점 u를 중심으로 하여 정점u 인접한 정점들을 한개의 연결 리스트로 표현함
- G는 n개의 연결리스트로 표현
- G의 간선들이 비용을 가지는 경우 연결 리스트상의 각 노드에 비용을 저장할 영역을 할당하면 된다. 이때 각 연결 리스트의 첫 번째 노드를 지시하는 포인터 배열이 추가로 필요함

이와 같은 인접 리스트는 아래그림 에서 보여 주는 것과 같이 순차적인 표현이나 연결 리스트 표현으로 나타낼 수 있으며, (a) 그래프 G (b) G에 대한 순차적인 인접 리스(c) G에 대한 연결 인접 리스트정점 vertex link

* 그래프 G에 대한 인접 리스트 표현



vertex	degree	인접 list
1	3	2 3 4
2	2	3 4
3	1	5
4	2	3 5
5	1	1



학습내용2 : 그래프의 순회

1. 깊이 우선 탐색

무 방향 그래프에서 어떤 정점에 대하여 검색이 끝나면 인접한 정점 중 검색하지 않은 정점을 찾아가 다시 검색하는 방법으로, 무방향 그래프 G(V, E)를 구성하는 V(G)에 속하는 모든 정점들을 방문하는 방법을 그래프의 운행이라 하며, 스택을 이용하는 깊이 우선 검색(DFS : Depth First Search)과 큐를 이용하는 너비 우선 검색(BFS : Breadth First Search) 운행 방법이 있다.

① 탐색 과정
㉠ DFS는 정점의 방문시 최근의 주변정점을 먼저 방문하는 순회방법

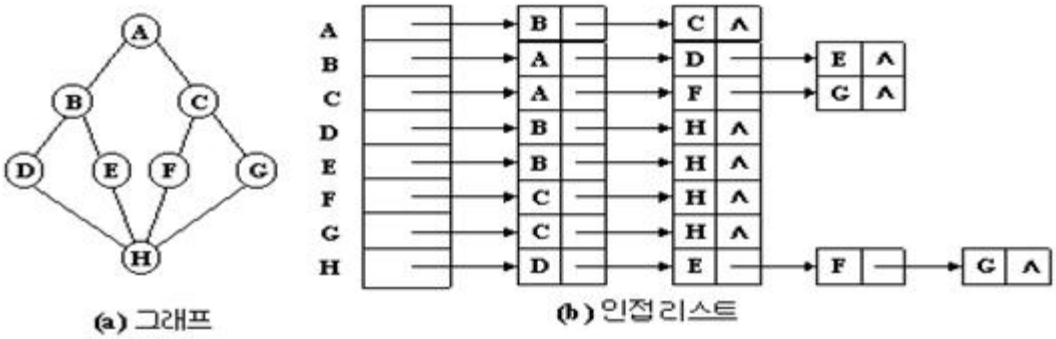
㉡ 과정

ⓐ 특정 vertex를 시점으로 선택한다.
ⓑ 선택된 vertex에 "방문"표시를 한다.
ⓒ 선택된 vertex에 연결된 여러 vertex들을 검사한다.
ⓓ 검사한 vertex들 중에 아직 방문하지 않은 vertex가 있으면 그 vertex를 새롭게 선택하고 원래 vertex는 스택에 넣는다.
ⓔ 검사한 vertex들 중에 방문하지 않은 vertex가 하나도 없으면 최종적으로 삽입된 vertex를 꺼내서 새롭게 선택한다.
ⓕ stack이 빌 때까지 2-5의 과정을 반복한다.

(1)-(6)의 과정이 끝나면 모든 vertex들을 한번씩 방문하게 된다.

② 깊이우선 탐색의 성능
- 시간 복잡도 $O(|V|^2)$

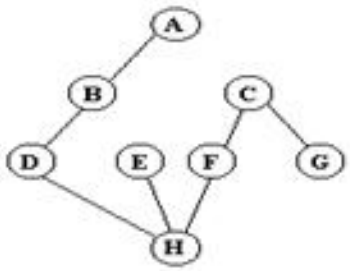
③ 깊이 우선 탐색 예제



④ 깊이우선 탐색 예제 해설

㉠ 먼저 정점 A에서 시작하여 정점 A에 인접한 정점은 정점 B와 C이다.
→ 두 정점 중 정점 B 를 방문한다.
㉡ 정점 B 에 인접한 정점은 정점 A 와 D, E 이다.
→ 정점 A 는 이미 방문했으므로 정점 D, E 중 정점 D 를 방문한다.
㉢ 정점 D 에 인접한 정점은 정점 B 와 H 이다.
→ 정점 B 는 이미 방문했으므로 정점 H 를 방문한다.
㉣ 정점 H 에 인접한 정점은 정점 D, E, F, G 이다.
→ 정점 D 는 이미 방문했으므로 정점 E, F, G 중 E 를 먼저 방문함
㉤ 정점 E 에 인접한 정점은 B 와 H 이다.
→ 정점 B 와 H 는 모두 방문했으므로 H 로 되돌아와
→ 정점 H 에 인접한 정점 D, E, F, G 중 방문하지 않은 정점 F 와 G 중
→ 정점 F 를 방문한다.
㉥ 정점 F 에 인접한 정점은 C 와 H 이다.
→ 정점 H 는 이미 방문했으므로 정점 C 를 방문한다.
㉦ 정점에 인접한 정점은 A, F, G 이다.
→ 정점 A 와 F 는 이미 방문했으므로 정점 G 를 방문한다.
◎모두 방문했으므로 DFS 를 종료한다.
⇒ 이렇게 방문한 정점을 모두 연결하여 그래프를 그리면 그림 6.12와 같음
→ 방문하는 노드 순서 : A B D H E F C G

* [그림] 그래프의 깊이 우선 탐색



2. 너비 우선 탐색

무방향성 그래프에서 어떤 정점을 검색하고 그 정점에 인접한 모든 정점들을 검색한 후 이 정점에 인접한 모든 정점들을 검색하는 방법으로 Queue를 이용
- 가깝게 인접한 정점을 모두 방문한 후 그 다음으로 가깝게 인접한 정점을 방문하는 순 으로 진행
- 큐(queue) 이용

① 탐색 과정
무방향 그래프 G(V, E)에서 시작하며 정점V를 방문한 후 V에 인접한 아직 방문하지 않은 모든 정점들을 방문한 뒤, 다시 이 정점에 인접하면서 방문하지 않은 모든 정점들에 대해 너비 우선 검색을 반복적으로 수행한다.

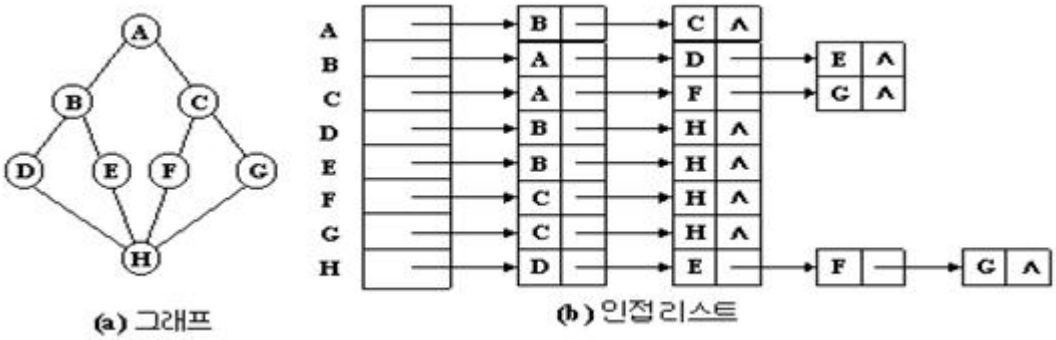
㉠ 특정 vertex를 시작점으로 선택한다.
㉡ 선택된 vertex에 "방문" 표시를 한다.
㉢ 선택된 vertex에 연결된 여러 vertex들을 검사하여 미방문 vertex들을 큐에 넣는다.
㉣ 큐의 front에서 하나의 vertex를 꺼내어 새롭게 선택한다.
㉤ 큐가 빌 때까지 2-4의 과정을 반복한다.

㉠-㉤의 과정을 끝내면 모든 vertex들을 한번씩 방문하게 된다.

무방향 그래프에서 너비 우선탐색과정	
단계1	시작정점V를 결정하여 방문한다.
단계2	정점V에 인접한 정점들 중에서 아직 방문하지 않은 모든 정점들을 방문하고 다시 이 정점에 대하여 인접하여 방문하지 않은 모든 정점에 대해 BFS방식으로 반복하여 방문한다.
단계3	더 이상 방문할 정점이 존재하지 않으면 방문을 종료한다.

② 성능
- 큐 작업에 소요되는 총 시간 O(|V|)
- 인접리스트의 모든 원소가 조사되는 시간 O(|V|)
- 총 시간 복잡도 : O(|V|+|E|)

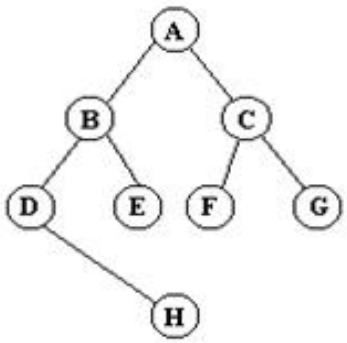
③ 예제
다음 그림의 그래프를 이용하여 너비우선탐색 방법



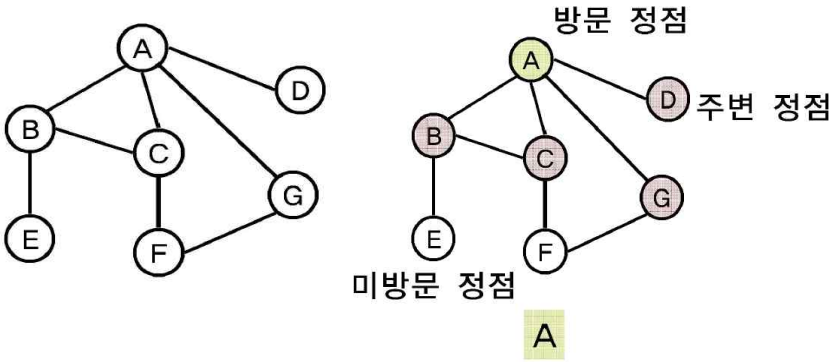
④ 해설

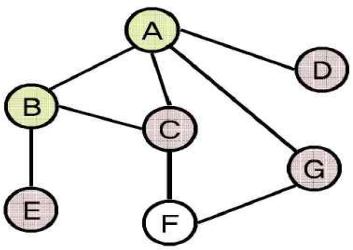
㉠ 먼저 정점 A에서 시작하여 정점 A에 인접한 B와 C를 차례로 방문한다.
㉡ 다음 정점 B에 인접한 정점 A, D, E 중 정점 A는 이미 방문했으므로 → 정점 D와 E를 차례로 방문하고,
㉢ 정점 C에 인접한 정점 A, F, G 중 정점 A는 이미 방문했으므로 → 정점 F와 G를 차례로 방문한다.
㉣ 다음 정점 D에 인접한 정점 B와 H 중 정점 B는 이미 방문했으므로 → 정점 H를 방문한다.
㉤ 모든 정점을 방문했으므로 BFS를 종료한다.
⇒ 이렇게 방문한 정점을 모두 연결하여 그래프를 그리면 그림 6.13와 같음
→ 방문하는 노드 순서 : A, B, C, D, E, F, G, H

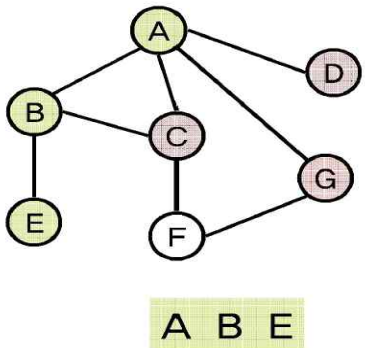
* [그림] 그래프의 너비 우선 탐색

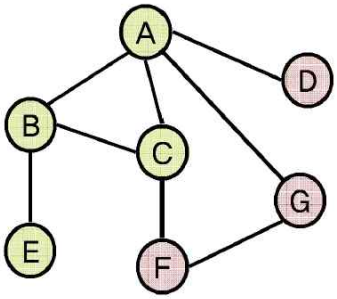


3. 깊이 우선 검색(DFS : Depth First Search)의 예 (1)

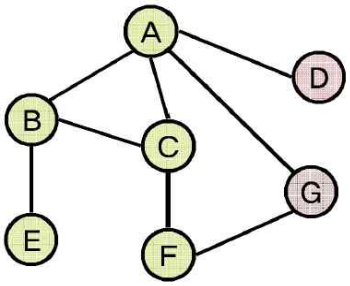




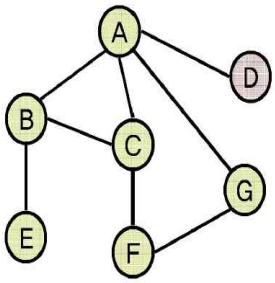




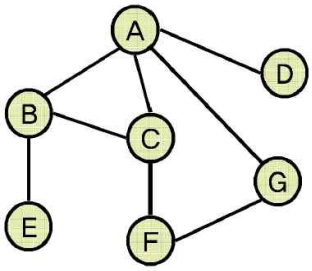
A B E C



A B E C F

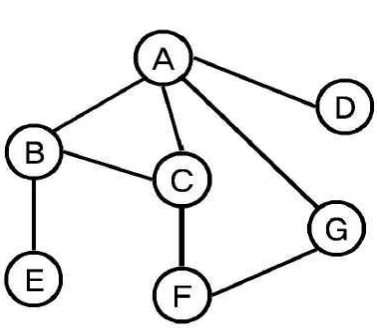


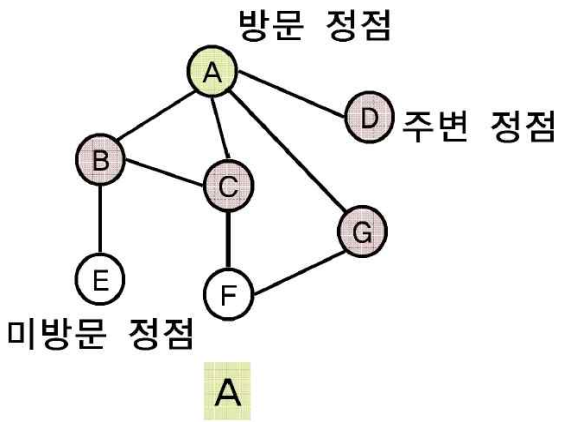
A B E C F G



A B E C F G D

4. 깊이 우선 검색(DFS : Depth First Search)의 예 (2)





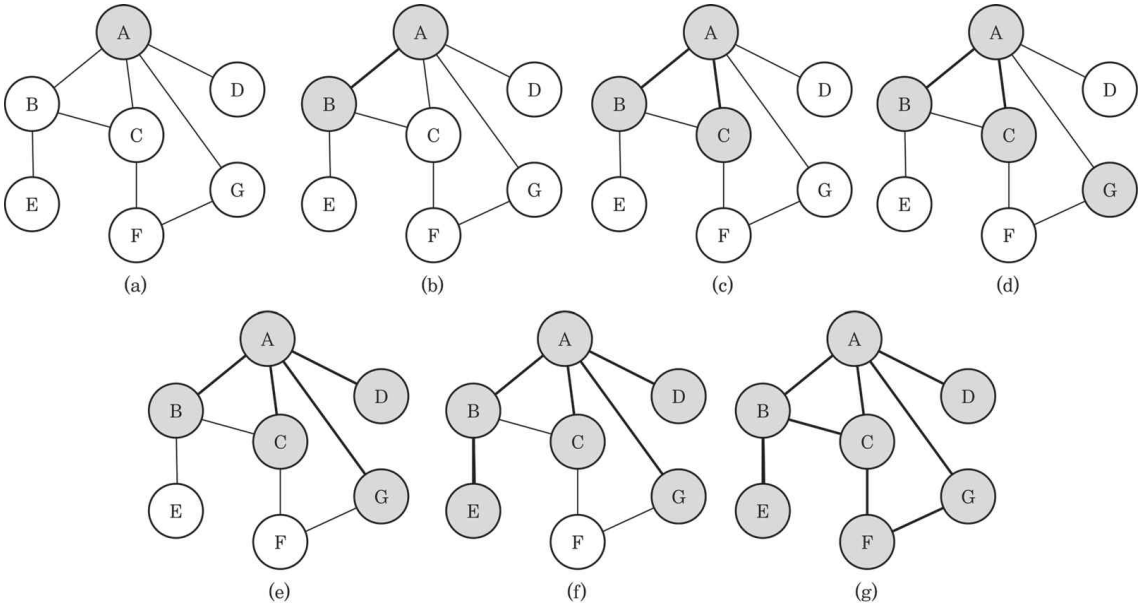
A

깊이 우선 검색(DFS)→ 방문하는 노드 순서 :A, G, F, C, B, E, D

5. 너비 우선 탐색(BFS ; Breach First Search)

우선 가깝게 인접한 정점을 모두 방문한 후 그 다음으로 가깝게 인접한 정점을 방문하는 순으로 진행한다(주변 정점 중에서 오래된 것부터 방문).

* 너비 우선 탐색 과정 : A B C G D E F

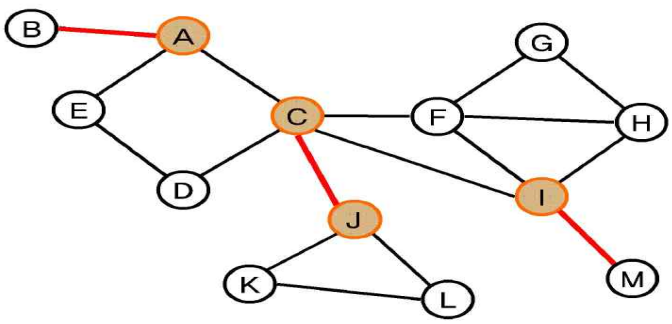


6. 깊이 우선 탐색과 너비 우선 탐색의 비교

구분	깊이 우선탐색(DFS)	너비 우선탐색(BFS)
방문점검	가장 최근에 나타난 주변 정점을 선택	가장 오래된 주변정점을 선택
특성	스택사용	큐 사용

학습내용3 : 그래프의 연결성

그래프에서 두 정점들 간에 경로가 존재하면 연결(connected)되어 있다고 함



접합점, 다리 , 이중 연결 성분

1. 연결성분

모든 정점 쌍 간에 경로가 존재하는 그래프를 연결된 그래프라고 하며, 연결성분이라고 하는 것은 연결된 최대의 부분그래프를 말한다.(너비 우선 탐색, 깊이 우선 탐색, 합-찾기 알고리즘 등을 통하여 구현함) 연결성분의 개수는 main 함수에서 dfs의 호출횟수에 의해 구한다.

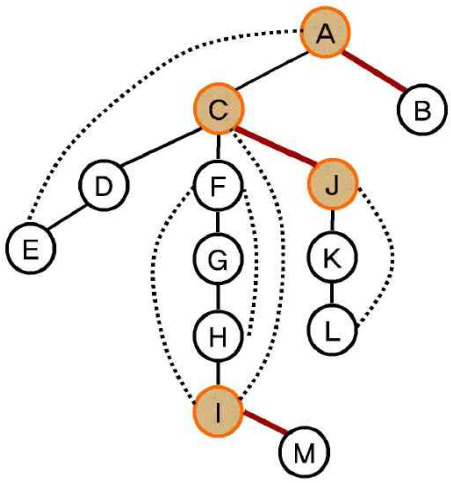
2. 이중 연결성(connectivity)

① 접합점(articulation point): 제거하면 G의 연결이 끊어지는 정점을 말한다.
② 다리(bridge): 제거하면 G의 연결이 끊어지는 간선을 의미한다.
③ 이중연결그래프(biconnected graph): 모든 정점 쌍 간에 둘 이상의 경로가 존재하는 그래프이다.
④ 이중연결성분(biconnected component): 모든 정점 쌍 간에 둘 이상의 경로가 존재하는 최대의 부분 그래프이다.

3. 접합점의 정리

① 깊이우선 나무에서 뿌리에 둘 이상의 자손이 있으면 뿌리는 접합점이다.
② 뿌리가 아닌 정점 u의 경우 u의 각 자식 v에 대하여 v와 그 자손 모두에서 점선 링크가 없으면 u는 접합점이다.
③ 접합점은 깊이우선탐색을 행하여 구할 수 있다.
④ 접합점을 구하는 알고리즘의 시간복잡도와 깊이우선 탐색의 시간복잡도의 크기 관계는 같다.

* 깊이우선탐색을 이용하여 깊이우선나무를 만든 예



다리의 필요 충분 조건

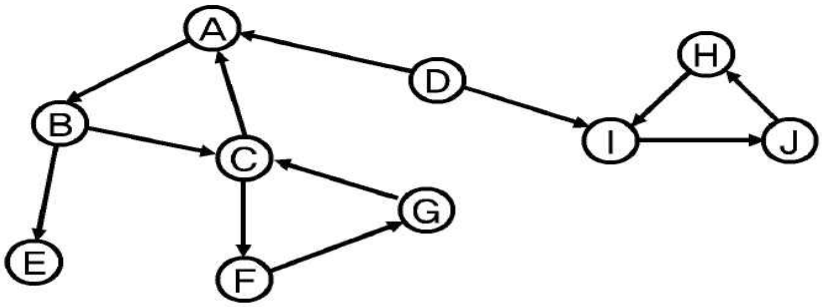
* 간선(u, v)가 다리
- (u, v)가 사이클에 포함되지 않음
- 접합점 u에 부수된 간선으로서 v를 포함한 아래 정점들에서 u를 포함한 아래 정점들에서 u를 포함한 위쪽 정점들에 연결되어 있는 링크가 없어야 한다.

4. 강 연결 성분(strongly connected component)

① 강 연결성분(strongly connected component): 방향 그래프에 대하여 성립하는 것으로 방향 그래프에서 서로 접근할 수 있는 방향경로가 모든 정점 쌍 간에 존재하는 최대의 부분 그래프로, 원래의 그래프와 역 그래프(원래의 그래프에서 간선의 방향을 역으로 하여 구하는 그래프)에 대하여 깊이 우선 탐색을 하여 구할 수 있다.

② 강 연결 그래프 : 강 연결 성분이 하나 이상 존재하는 그래프

③ 시간 복잡도 : O(|V|+|E|)



{A,B,C,F,G}, {D}, {E}, {H,I,J}

* 강 연결 성분 알고리즘

- G=(V, E)

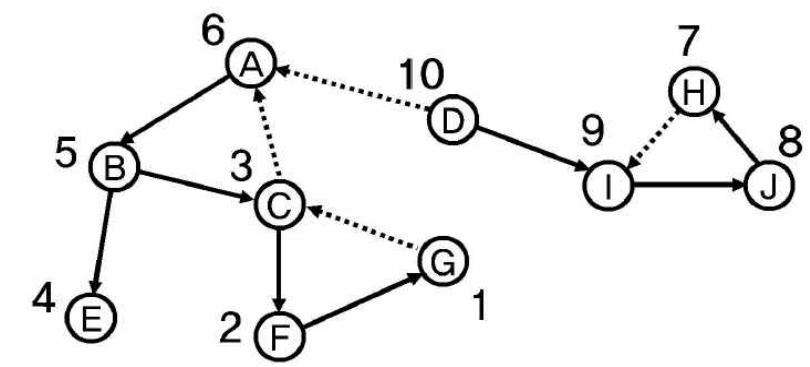
- E^R={⟨u, v⟩ : ⟨v, u⟩∈E}
→ 간선의 방향을 반대로 한 간선의 집합.

- G^R=(v, E^R) 역그래프
→ 정점과 반대 방향의 간선의 집합.

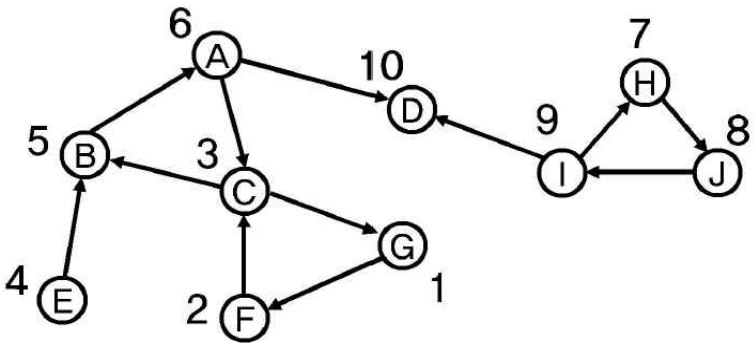
- dfs_visit(v) 수정
→ flag[v] : 방문을 완료한 순서 번호 기록

[단계 1] dfs(G)(깊이우선)를 수행하여 방문 완료 순서를 구함.
[단계 2] G^R 을 구함
[단계 3] dfs(G^R)을 하는데 dfs의 줄 5의 for 루프에서 방문 완료 순서의 역순으로 dfs_visit을 호출하도록 함.
[단계 4] 단계 3의 결과로 만들어지는 깊이 우선 숲의 각 나무에 속하는 정점들이 강 연결 성분임.

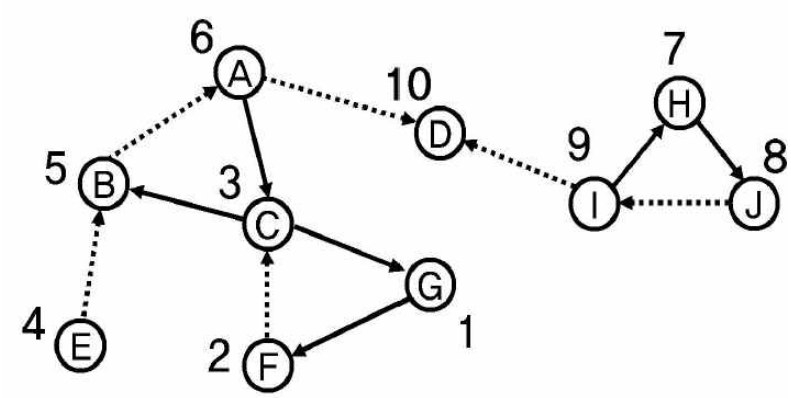
[단계 1] dfs(G)를 수행하여 방문 완료 순서를 구함.



[단계 2] G^R 을 구함



* (G^R)에 대한 깊이 우선 숲

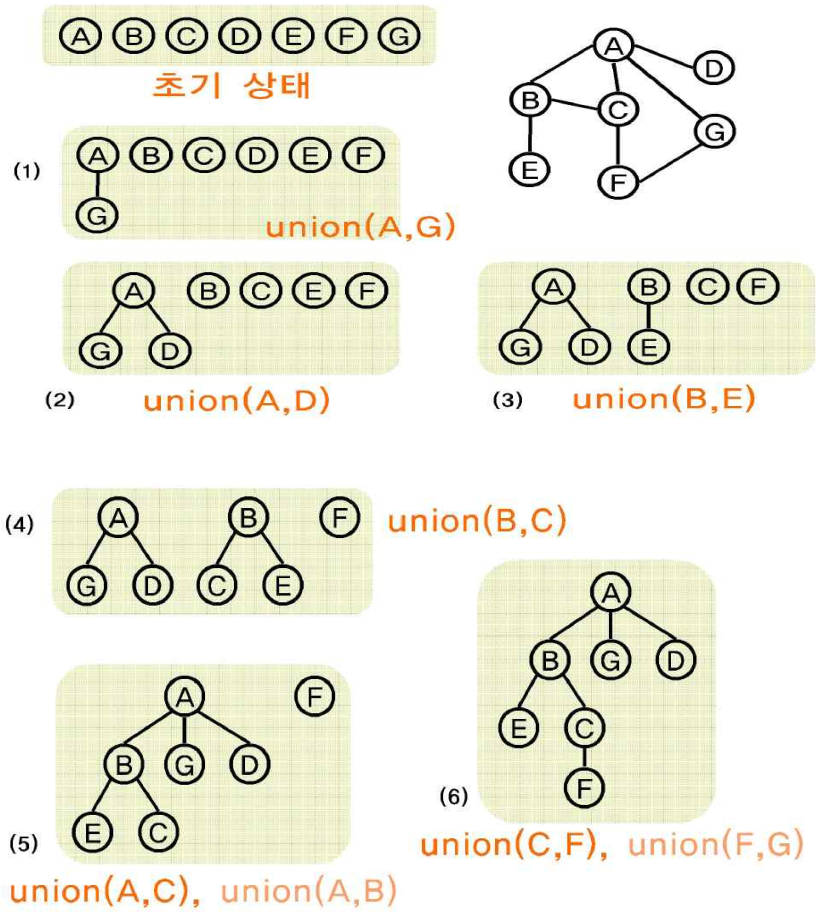


5. 합-찾기 문제

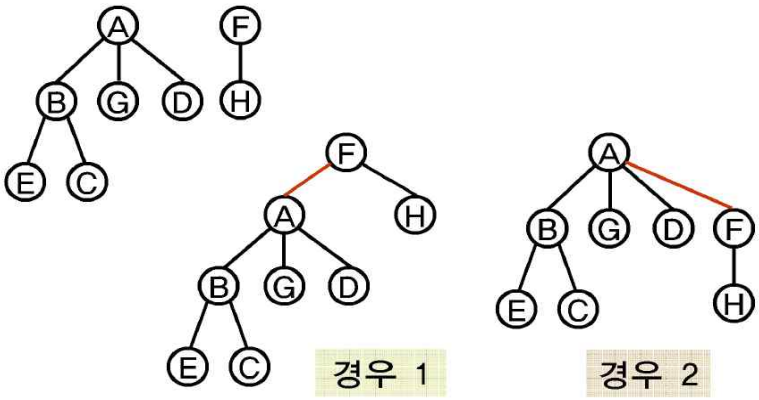
합은 두 집합의 합집합을 구하는 연산이며, 찾기 연산은 어떤 원소가 속하는 집합을 찾아내는 연산으로 연결성분 문제 또는 사이클의 형성 여부를 조사하는 데 사용 할 수 있다.

① 두 연산 union(u,v), find(u)가 행하는 작업
init_S(x) : 원소 x만을 찾는 새 집합을 생성.
union(u, v): u와 v가 속한 집합을 합친다.
find(u): u가 속한 집합을 찾는다.

* 합 찾기 알고리즘
1) 그래프에 간선이 동적으로 추가된다면 합-찾기 알고리즘을 사용.
2) 그래프가 정적이면 그래프 순회 알고리즘을 사용.
3) 두 정점이 같은 연결 성분에 속하는 지 여부(정점 x와 y간을 연결하는 경로의 존재 여부)



union(C, H)에 대한 나무의 형태

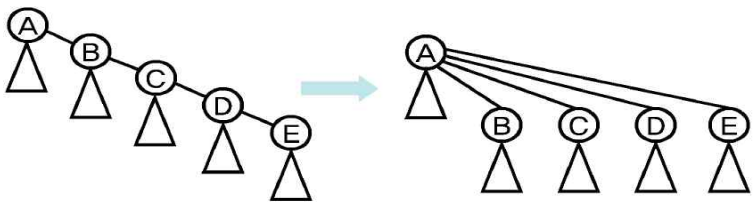


경우 1 경우 2

* 최악의 경우
- 경사나무 : find()에서 많은 시간이 소요 된다.

* 뿌리를 직접 연결하여 나무의 높이를 낮춘다.
- 등급(rank) : 각 노드 높이의 상한치
- 등급이 낮은 뿌리를 등급이 높은 뿌리 밑에 연결.

② 합-찾기 문제에서 등급과 경로 압축을 도입하여 얻는 이점은 나무의 높이가 낮아져 결과적으로 시간복잡도가 낮아진다. 이상적으로는 모든 노드가 뿌리에 연결되는 것이 좋다. 이것에 근접한 기법으로 경로 압축(path compression)이라는 것을 find 연산에서 적용하는데 이는 find(u)에서 뿌리를 찾은 후 뿌리에서 u까지의 경로 상에 있는 모든 노드가 뿌리에 직접 연결되도록 바꾸는 것이다.



find(E)시 경로 압축

* 합-찾기 연산의 시간 복잡도
◎ 노드의 개수 n개, 적용 연산의 횟수 m번
◎ 등급에 의한 합만을 행하는 경우 : O(mlogn)
◎ 등급에 의한 합과 경로 압축을 행하는 경우 : $O(m \cdot \mathcal{L}(m.n)) \rightarrow O(m)$

【학습정리】

1. 그래프 G=(V,E) : 정점의 집합 V와 간선의 집합 E의 집합
- 기본적인 용어 : 가중 그래프, 무방향 그래프, 방향 그래프, 네트워크, 부수, 인접, 경로, 경로의 길이, 단순경로, 연결 그래프, 부분 그래프, 연결 성분, 사이클, 신장 나무, 완전 그래프

2. 그래프의 표현
- 정점들의 관계를 나타내는 간선의 집합 E를 표현하는 것
- 인접 행렬에 의한 표현: |V|*|V| 행렬 사용, 조밀한 그래프에 적합
- 인접 리스트에 의한 표현: 연결 리스트 사용, 성긴 그래프 표현에 적합

3. 그래프 순회
- 그래프의 모든 정점을 체계적으로 방문하는 것 (깊이 우선 탐색, 너비 우선 탐색)
- 깊이 우선 탐색
  → 최근의 주변 정점을 우선적으로 방문하는 방법
  → 스택으로 구현하는 것이 적절,
  → 인접리스트 O(|V|+|E|), 인접행렬 $O(|V|^2)$
- 너비 우선 탐색
  → 주변 정점 중에서 가장 오래된 것부터 먼저 방문하는 방법
  → 큐로 구현하면 적절
  → 인접리스트 O(|V|+|E|)
- 위상 정렬 : 무사이클 방향 그래프(dag)의 방향 간선이 한 방향으로만 향하도록 정점을 나열하는 것, 깊이 우선 탐색 시에 방문을 완료한 정점을 역순으로 나열하여 구함

4. 그래프의 연결성
- 연결 성분
    → 모든 정점쌍 간에 경로가 존재하는 최대의 부분 그래프
- 접합점, 다리, 이중연결성분이란?
    → 접합점: 제거하면 G의 연결이 끊어지는 정점
    → 다리: 제거하면 G의 연결이 끊어지는 간선
    → 이중 연결 성분: 모든 정점쌍 간에 둘 이상의 경로가 존재하는 최대의 부분 그래프
- 강연결 성분
    → 강연결 성분은 방향 그래프에 대하여 성립하는 것으로서 방향 그래프에서서로 접근할 수 있는 방향 경로가 모든 정점쌍 간에 존재하는 최대의 부분 그래프이다.
    → 원래의 그래프와 역 그래프(원래의 그래프에서 간선의 방향을 역으로 하여 구한 그래프)에 대하여 깊이 우선 탐색을 하여 구할 수 있다. (이때 걸리는 시간은 O(|V|+|E|)이다.)

5. 합-찾기 문제
- 합은 두 집합의 합집합을 구하는 연산이며, 찾기 연산은 어떤 원소가 속하는 집합을 찾아내는 연산이다.
- 연결성분 문제, 또는 사이클의 형성 여부를 조사하는 데 사용할 수 있다.