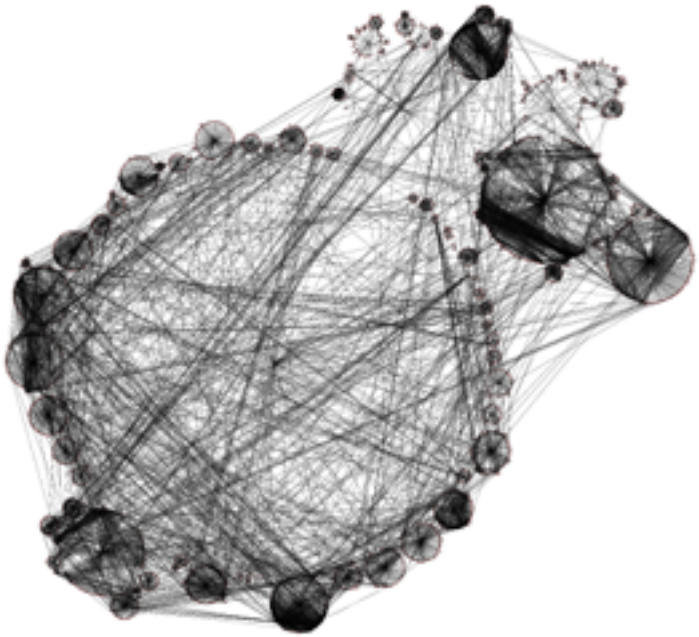


제6장

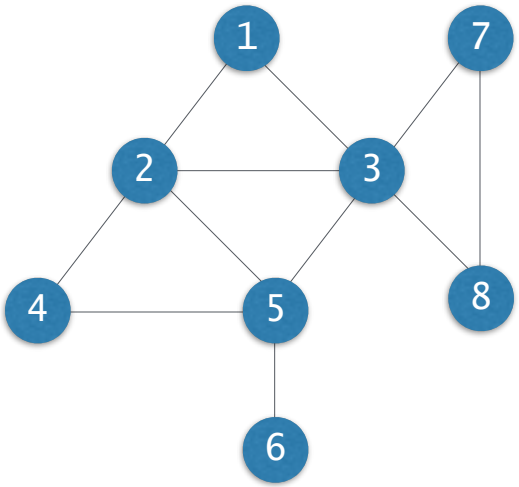
그래프 알고리즘

(Graph Algorithms)

개념과 표현
Concepts and Representations

그래프 (Graph)

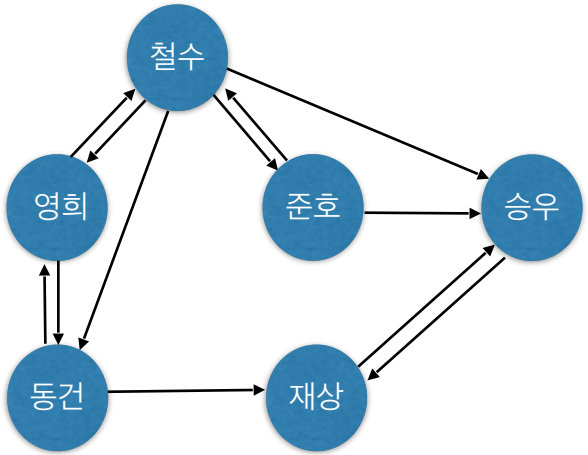
- (무방향) 그래프 G=(V,E)
  - V : 노드(node) 혹은 정점(vertex)
  - E : 노드쌍을 연결하는 에지(edge) 혹은 링크(link)
  - 개체(object)들 간의 이진관계를 표현
  - n=|V|, m=|E|



V={1,2,3,4,5,6,7,8}
E={(1,2),(1,3),(2,3),...,(7,8)}
n=8
m=11

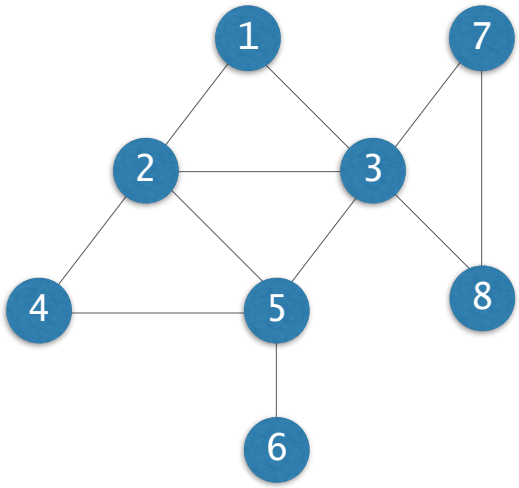
방향 그래프와 가중치 그래프

- 방향그래프(Directed Graph) G=(V,E)
  - 에지 (u,v)는 u로부터 v로의 방향을 가짐



- 가중치(weighted) 그래프
  - 에지마다 가중치(weight)가 지정

- 인접행렬 (adjacency matrix)
  - n×n 행렬 $A = (a_{ij}),\ \text{where } a_{ij} = \begin{cases} 1 & \text{if } (i,j) \in E, \\ 0 & \text{otherwise.} \end{cases}$



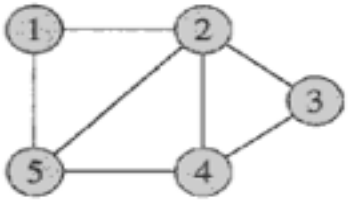
	1	2	3	4	5	6	7	8
1	0	1	1	0	0	0	0	0
2	1	0	1	1	1	0	0	0
3	1	1	0	0	1	0	1	1
4	0	1	0	0	1	0	0	0
5	0	1	1	1	0	1	0	0
6	0	0	0	0	1	0	0	0
7	0	0	0	0	0	0	0	1
8	0	0	1	0	0	0	1	0

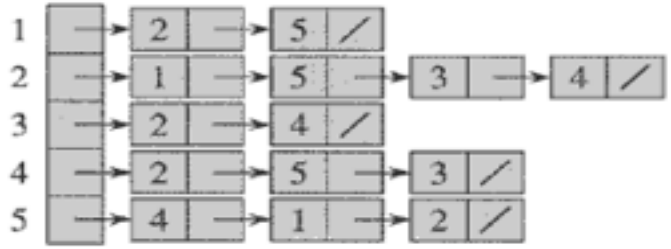
대칭행렬

저장 공간: $O(n^2)$
어떤 노드 v에 인접한 모든 노드 찾기: O(n) 시간
어떤 에지 (u,v)가 존재하는지 검사: O(1) 시간

그래프의 표현

- 인접리스트 (adjacency list)
  - 정점 집합을 표현하는 하나의 배열과
  - 각 정점마다 인접한 정점들의 연결 리스트



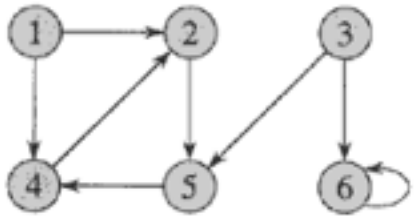


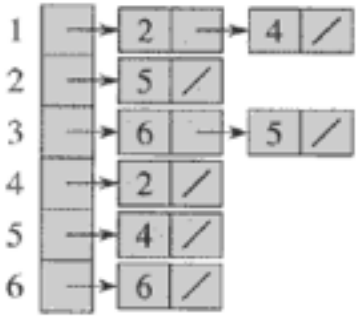
노드개수: 2m

저장 공간: O(n+m)
어떤 노드 v에 인접한 모든 노드 찾기: O(degree(v)) 시간
어떤 에지 (u,v)가 존재하는지 검사: O(degree(u)) 시간

방향그래프

- 인접행렬은 비대칭
- 인접 리스트는 m개의 노드를 가짐





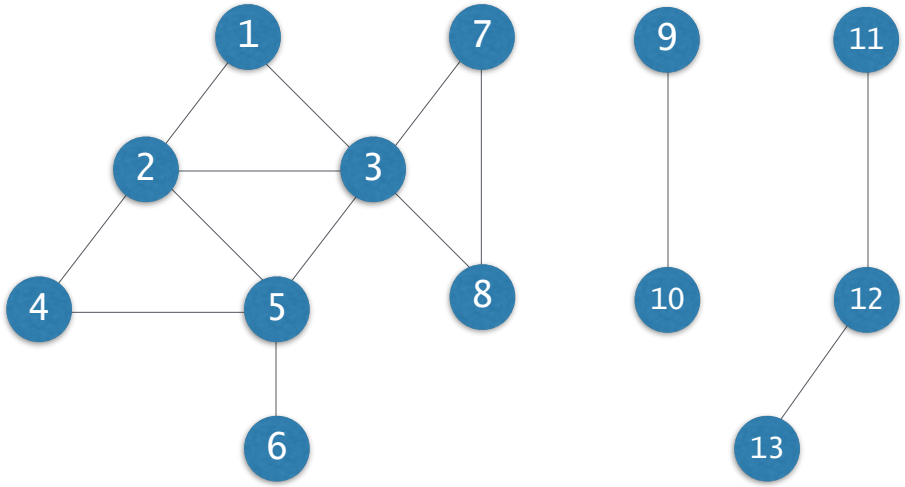
	1	2	3	4	5	6
1	0	1	0	1	0	0
2	0	0	0	0	1	0
3	0	0	0	0	1	1
4	0	1	0	0	0	0
5	0	0	0	1	0	0
6	0	0	0	0	0	1

가중치 그래프의 인접행렬 표현

- 에지의 존재를 나타내는 값으로 1대신 에지의 가중치를 저장
- 에지가 없을 때 혹은 대각선:
  - 특별히 정해진 규칙은 없으며, 그래프와 가중치가 의미하는 바에 따라서
  - 예: 가중치가 거리 혹은 비용을 의미하는 경우라면 에지가 없으면 ∞, 대각선은 0.
  - 예: 만약 가중치가 용량을 의미한다면 에지가 없을때 0, 대각선은 ∞

경로와 연결성

- 무방향 그래프 G=(V,E)에서 노드 u와 노드 v를 연결하는 경로(path)가 존재할 때 v와 u는 서로 연결되어 있다고 말함
- 노든 노드 쌍들이 서로 연결된 그래프를 연결된(connected) 그래프라고 한다.
- 연결요소 (connected component)



3개의 연결요소로 구성

그래프 순회
Graph Traversal

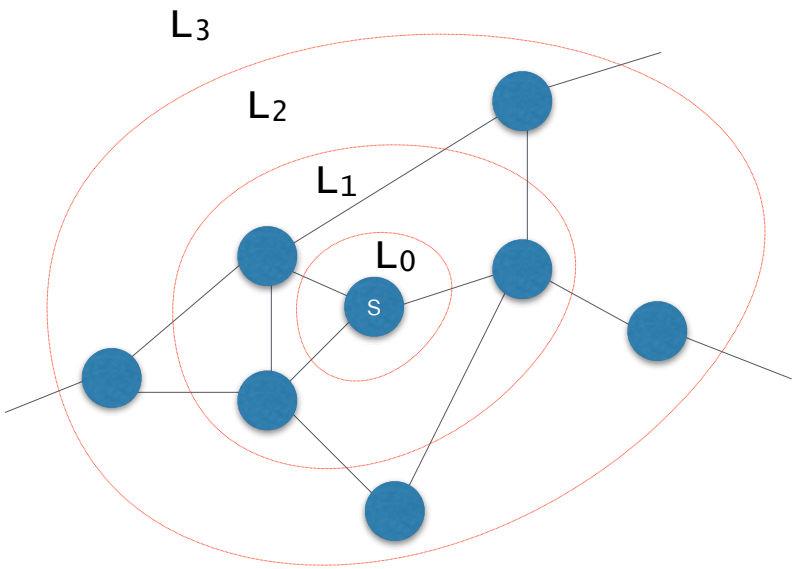
그래프 순회

- 순회(traversal)
  - 그래프의 모든 노드들을 방문하는 일
- 대표적 두 가지 방법
  - BFS (Breadth-First Search, 너비우선순회)
  - DFS (Depth-First Search, 깊이우선순회)

너비우선순회 (BFS)

- BFS 알고리즘은 다음 순서로 노드들을 방문
  - L_0 = {s}, 여기서 s는 출발 노드
  - L_1 = L_0의 모든 이웃 노드들
  - L_2 = L_1의 이웃들 중 L_0에 속하지 않는 노드들

  …
  - L_i = L_{i-1}의 이웃들 중 L_{i-2} 에 속하지 않는 노드들

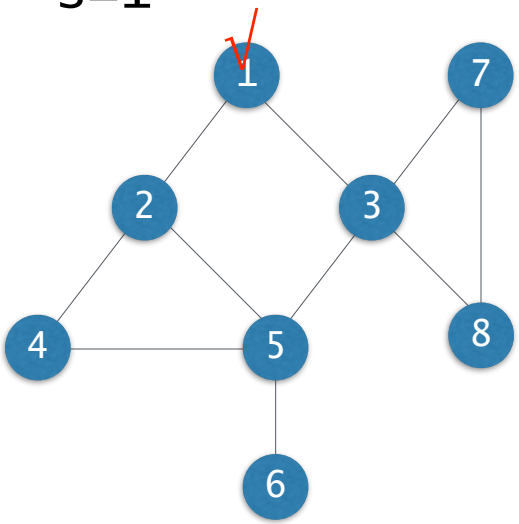


동심원 형태

큐를 이용한 너비우선순회

체크는 이미 방문된 노드라는 표시

s=1



1. check the start node;
2. insert the start node into the queue;

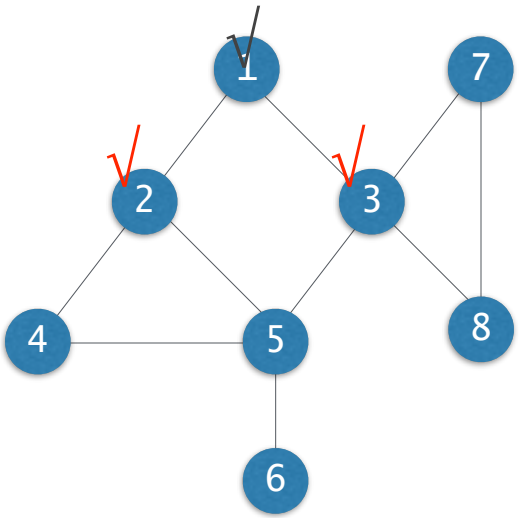
queue

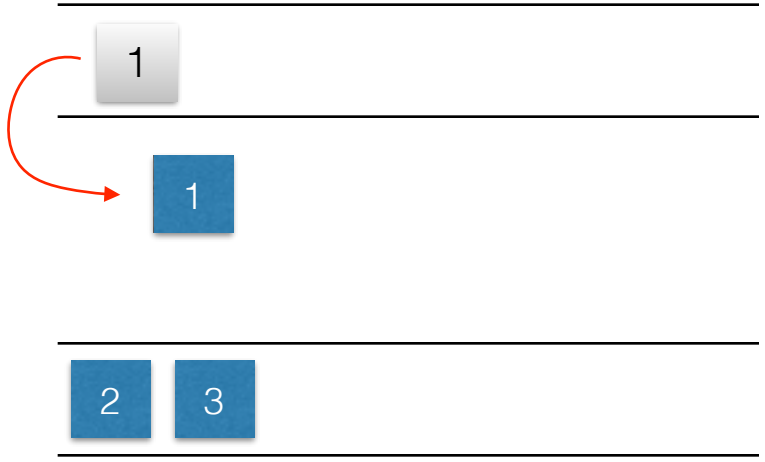
1

큐를 이용한 너비우선순회

```
while the queue is not empty do
   remove a node v from queue;
   for each unchecked neighbour w of v do
      check and insert w into the queue;
```

s=1

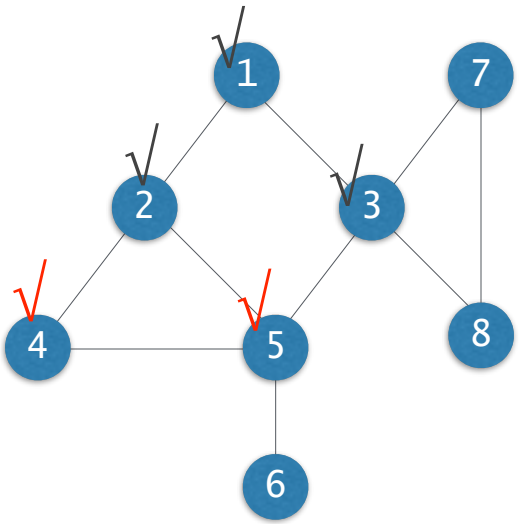


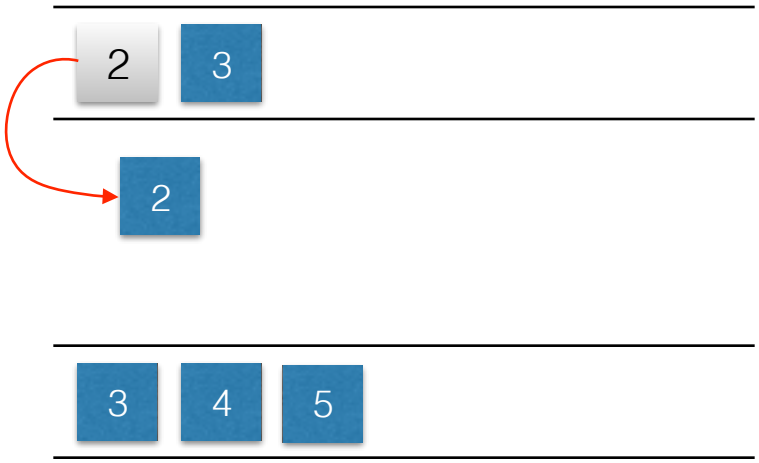


큐를 이용한 너비우선순회

```
while the queue is not empty do
   remove a node v from queue;
   for each unchecked neighbour w of v do
      check and insert w into the queue;
```

s=1

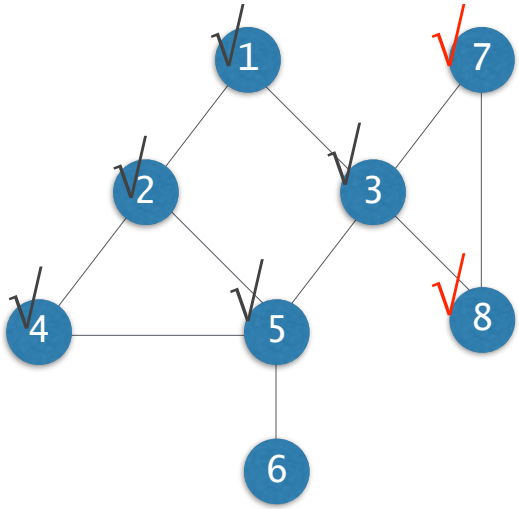


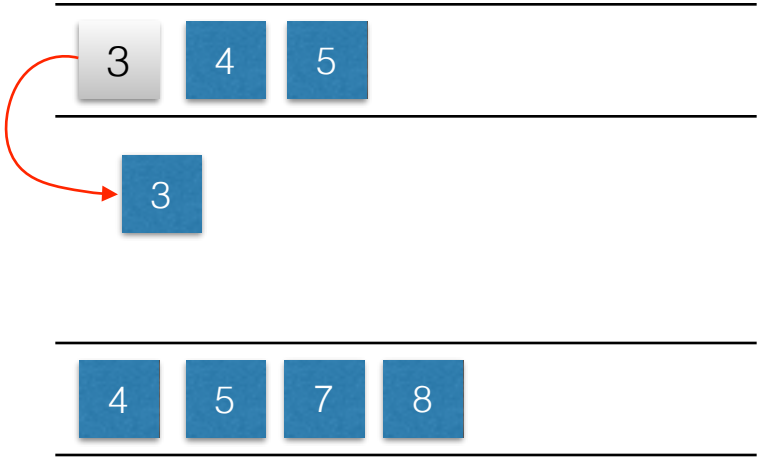


큐를 이용한 너비우선순회

```
while the queue is not empty do
   remove a node v from queue;
   for each unchecked neighbour w of v do
      check and insert w into the queue;
```

s=1

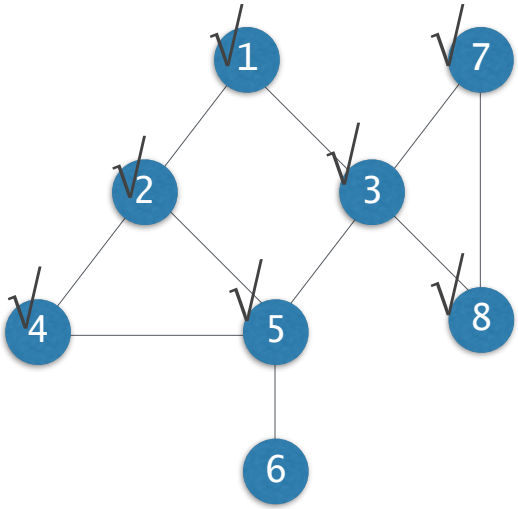


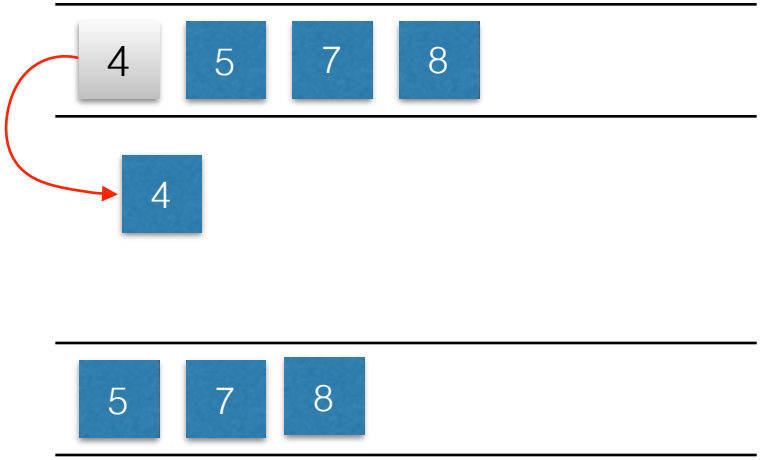


큐를 이용한 너비우선순회

```
while the queue is not empty do
   remove a node v from queue;
   for each unchecked neighbour w of v do
      check and insert w into the queue;
```

s=1

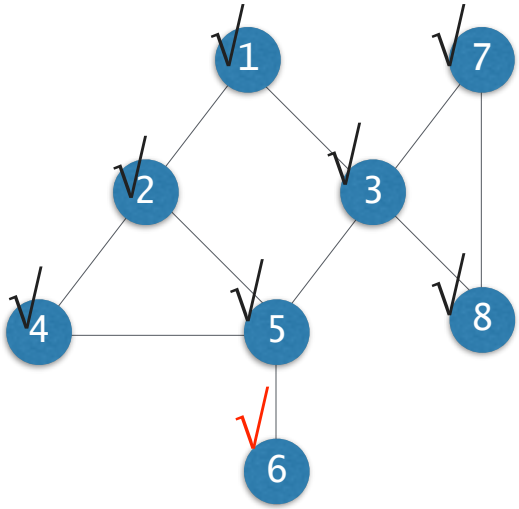


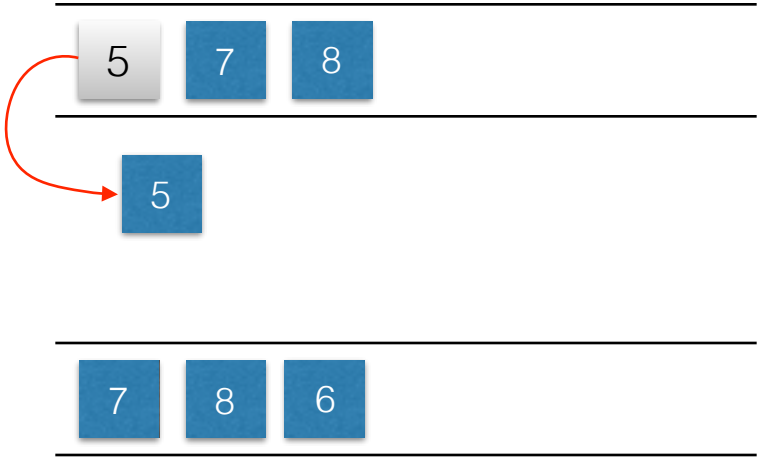


큐를 이용한 너비우선순회

```
while the queue is not empty do
   remove a node v from queue;
   for each unchecked neighbour w of v do
      check and insert w into the queue;
```

s=1

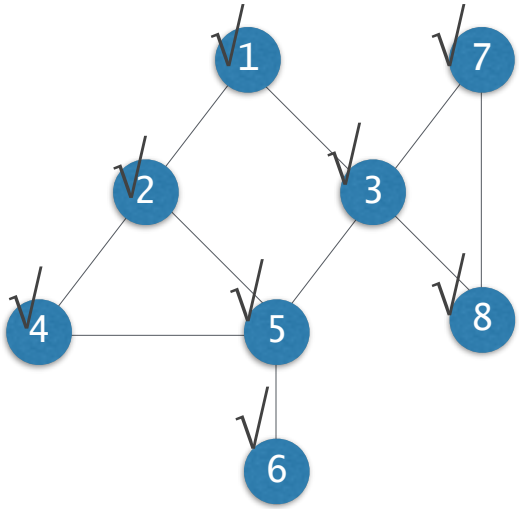


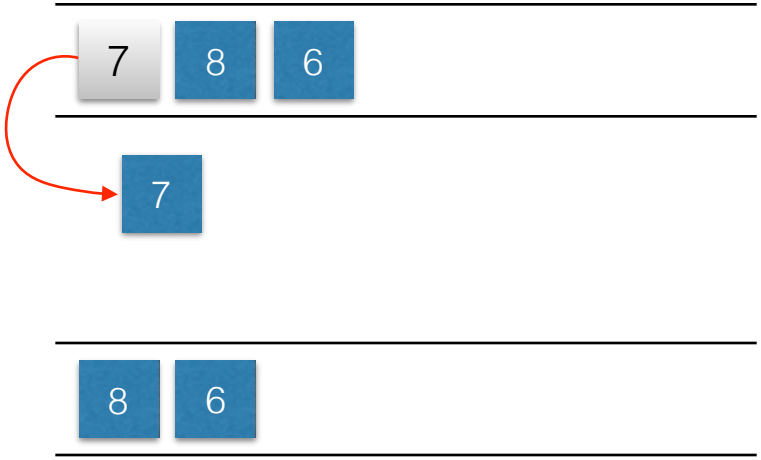


큐를 이용한 너비우선순회

```
while the queue is not empty do
   remove a node v from queue;
   for each unchecked neighbour w of v do
      check and insert w into the queue;
```

s=1

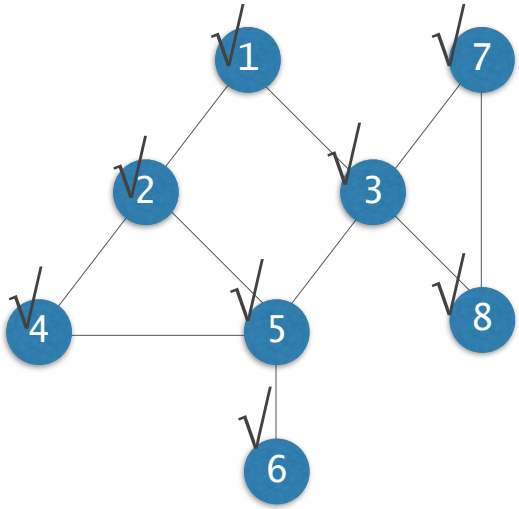


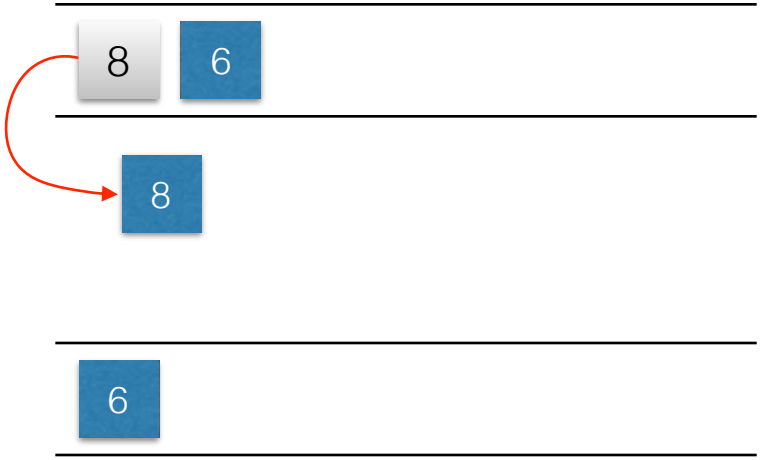


큐를 이용한 너비우선순회

```
while the queue is not empty do
   remove a node v from queue;
   for each unchecked neighbour w of v do
      check and insert w into the queue;
```

s=1

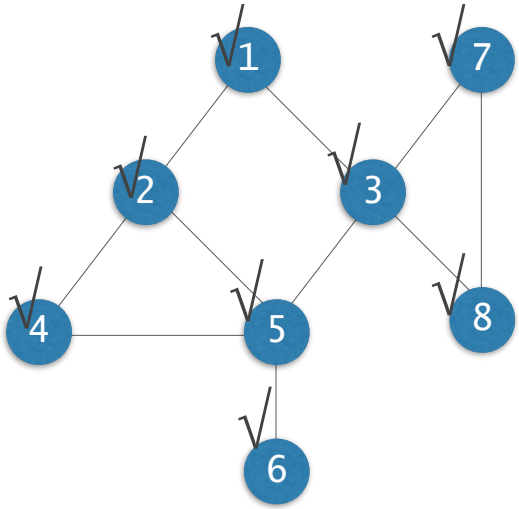


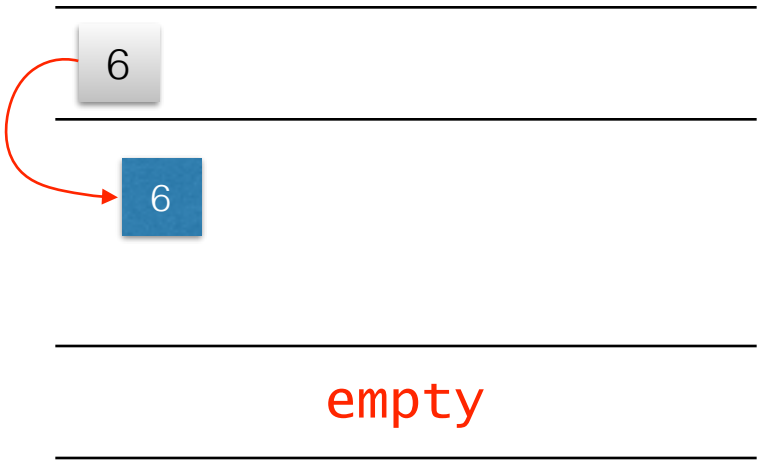


큐를 이용한 너비우선순회

```
while the queue is not empty do
   remove a node v from queue;
   for each unchecked neighbour w of v do
      check and insert w into the queue;
```

s=1





노드 방문 순서: 1, 2, 3, 4, 5, 7, 8, 6

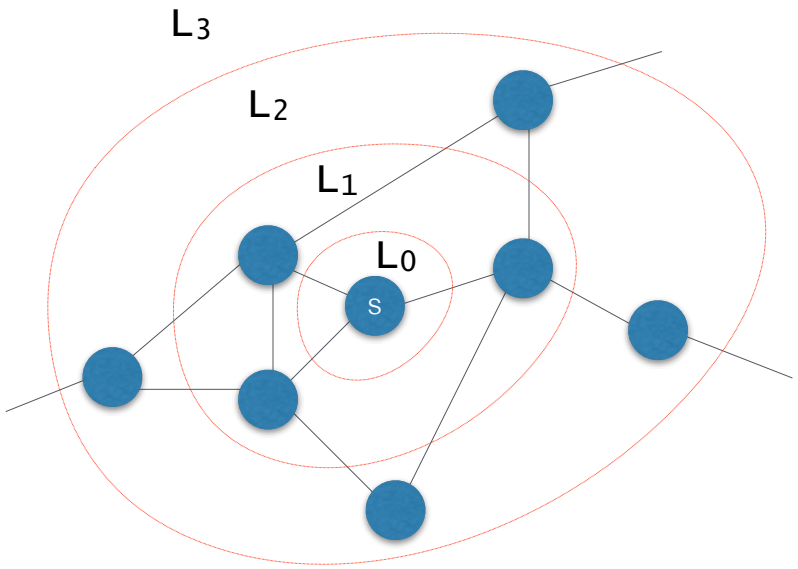
너비우선순회

그래프 G와 출발 노드 s

```
BFS(G,s)
  Q←∅;
  Enqueue(Q,s);
  while Q≠∅ do
    u ← Dequeue(Q)
    for each v adjacent to u do
      if v is unvisited then
         mark v as visited;
         Enqueue(Q,v);
      end.
    end.
  end.
```

BFS와 최단경로

- s에서 L_i에 속한 노드까지의 최단 경로의 길이는 i이다. 여기서 경로의 길이는 경로에 속한 에지의 개수를 의미한다.
- BFS를 하면서 각 노드에 대해서 최단 경로의 길이를 구할 수 있다.



동심원 형태

BFS와 최단경로

- 입력: 방향 혹은 무방향 그래프 G=(V,E), 그리고 출발노드 s∈V
- 출력: 모든 노드 v에 대해서
  - d[v] = s로 부터 v까지의 최단 경로의 길이(에지의 개수)
  - π[v] = s로부터 v까지의 최단경로상에서 v의 직전 노드(predecessor)

너비우선순회

```
BFS(G,s)
  Q←∅;
  d[s]←0;                  /* distance from s to s is 0 */
  π[s]←null;               /* no predecessor of s */
  Enqueue(Q,s);
  while Q≠∅ do
    u ← Dequeue(Q)
    for each v adjacent to u do
      if v is unvisited then
         mark v as visited;
         d[v]←d[u]+1;       /* distance to v */
         π[v]←u;            /* u is the predecessor of v */
         Enqueue(Q,v);
    end.
  end.
```

보통 모든 노드들에 대해서 d[v]를 -1로 초기화해두고, -1이면 unvisited, 아니면 visited로 판단한다.

O(n+m) with adjacent list

시간복잡도

```
BFS(G,s)
  Q←∅;
  for each node u do
    d[u] ← -1;
    π[u] ← null;
  end.
  d[s]←0; π[s]←null;
  Enqueue(Q,s);
  while Q≠∅ do
    u ← Dequeue(Q)
    for each v adjacent to u do
      if (d[v]==-1) then
         d[v]←d[u]+1;
         π[v]←u;
         Enqueue(Q,v);
    end.
  end.
```

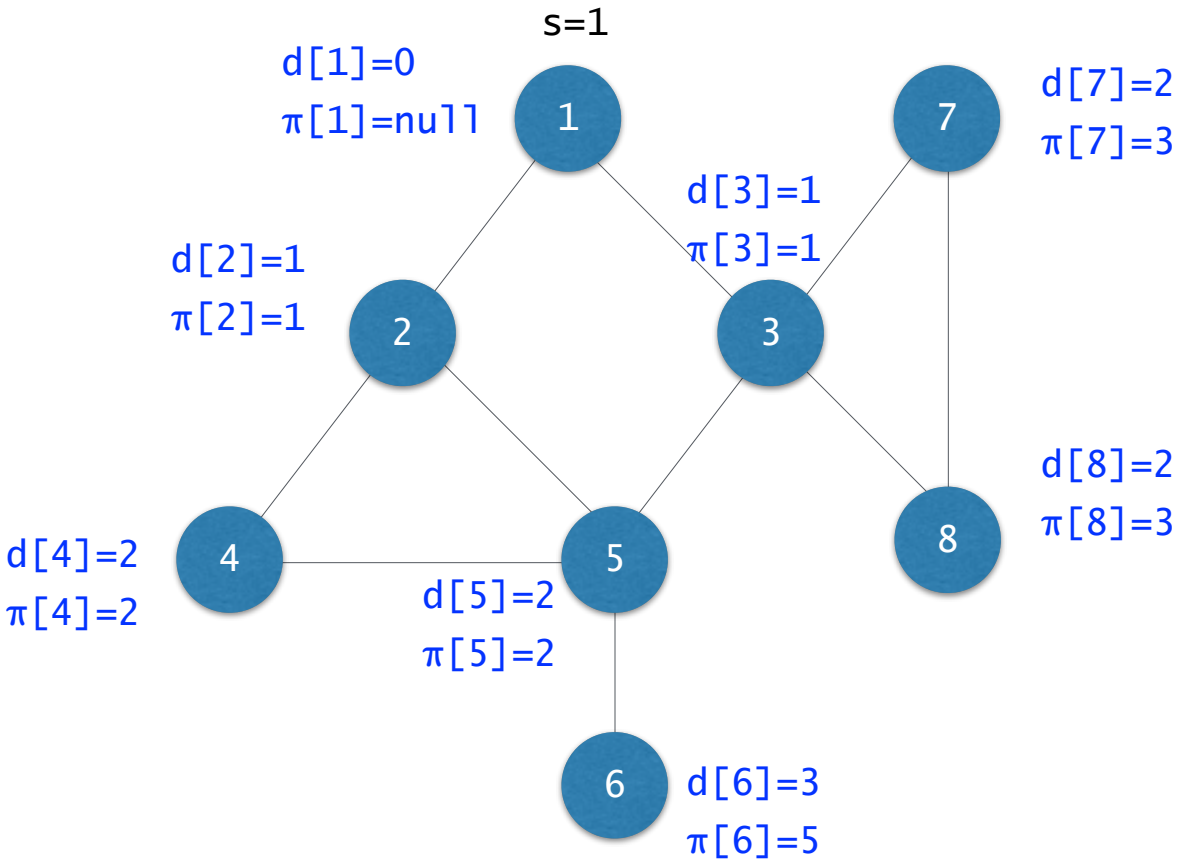
인접리스트로 구현할 경우 시간복잡도는

$\sum_{v} \text{degree}(v) = 2m$ 이므로 O(n+m)

while문을 한 번 돌 때마다 큐에서 하나를 꺼내므로 while문은 최대 n번 돈다.

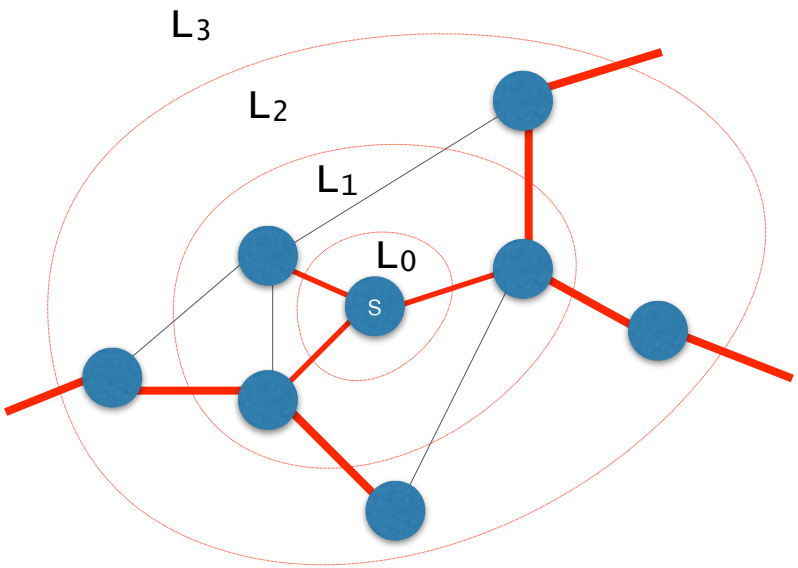
인접리스트로 구현할 경우 for문은 각 노드 v에 대해서 degree(v)번 돈다.

unchecked 노드만 queue에 들어갈 수 있으므로 어떤 노드도 큐에 두번 들어가지는 않는다.


s=1
d[1]=0
π[1]=null
1
7
d[7]=2
π[7]=3
d[3]=1
π[3]=1
d[2]=1
π[2]=1
2
3
d[8]=2
π[8]=3
d[4]=2
π[4]=2
4
5
8
d[5]=2
π[5]=2
6
d[6]=3
π[6]=5

BFS 트리

- 각 노드 v와 π[v]를 연결하는 에지들로 구성된 트리

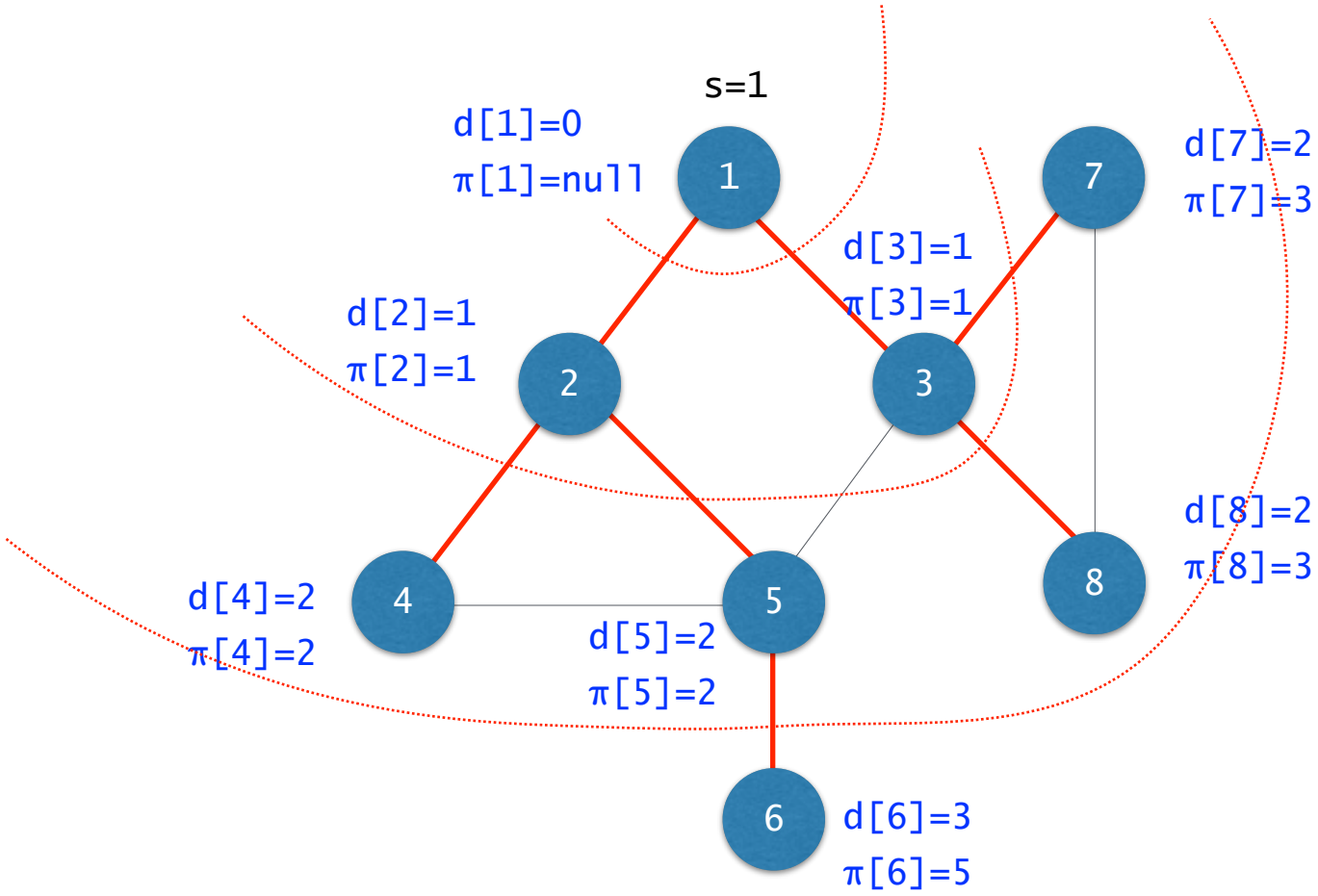


BFS 트리에서 s에서 v까지의 경로는 s에서 v까지 가는 최단경로

어떤 에지도 2개의 layer를 건너가지 않는다.
(동일 레이어의 노드를 연결하거나, 혹은 1개의 layer를 건너간다.)

BFS Tree

- 각 노드 v와 π[v]를 연결하는 에지들로 구성된 트리



너비우선순회: 최단 경로 출력하기

```
PRINT-PATH(G,s,v)     /* 출발점 s에서 노드 v까지의 경로 출력하기 */
  if v=s then
     print s;
  else if π[v]=null then
     print “no path from s to v exists”;
  else
     PRINT-PATH(G,s,π[v]);
     print v;
end.
```

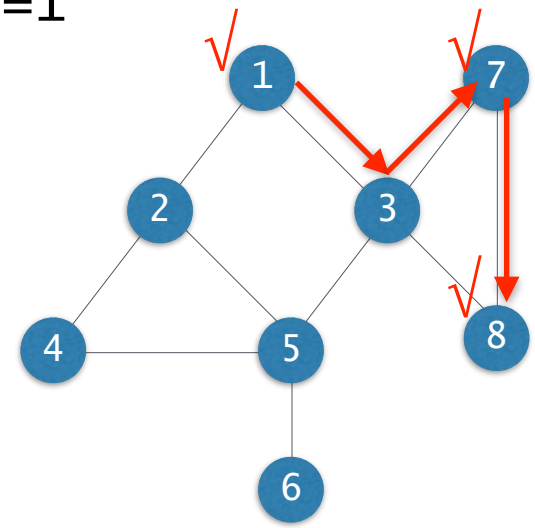
너비우선순회 (BFS)

- 그래프가 disconnected이거나 혹은 방향 그래프라면 BFS에 의해서 모든 노드가 방문되지 않을 수도 있음
- BFS를 반복하여 모든 노드 방문

```
BFS-ALL( G )
{
   while there exists unvisited node v
      BFS(G, v);
}
```

깊이우선순회 (DFS)

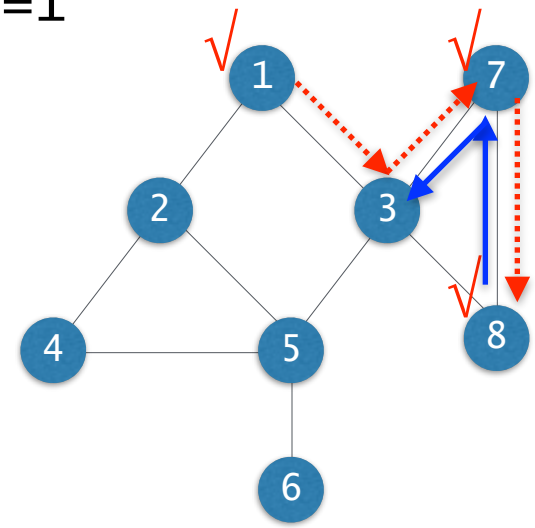
s=1



1. 출발점 s에서 시작한다.
2. 현재 노드를 visited로 mark하고 인접한 노드들 중 unvisited 노드가 존재하면 그 노드로 간다.
3. 2번을 계속 반복한다.

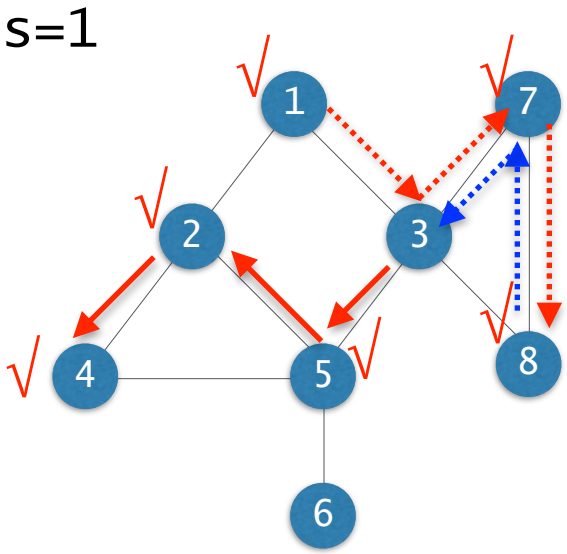
깊이우선순회 (DFS)

s=1



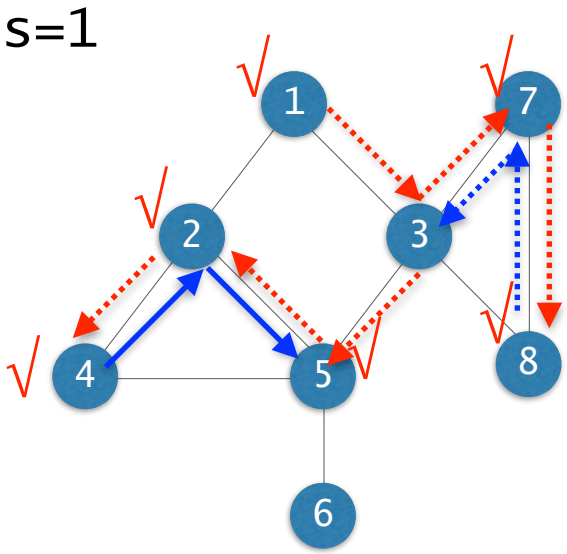
1. 출발점 s에서 시작한다.
2. 현재 노드를 visited로 mark하고 인접한 노드들 중 unvisited 노드가 존재하면 그 노드로 간다.
3. 2번을 계속 반복한다.
4. unvisited인 이웃 노드가 존재하지 않는 동안 계속해서 직전 노드로 되돌아간다.

깊이우선순회 (DFS)



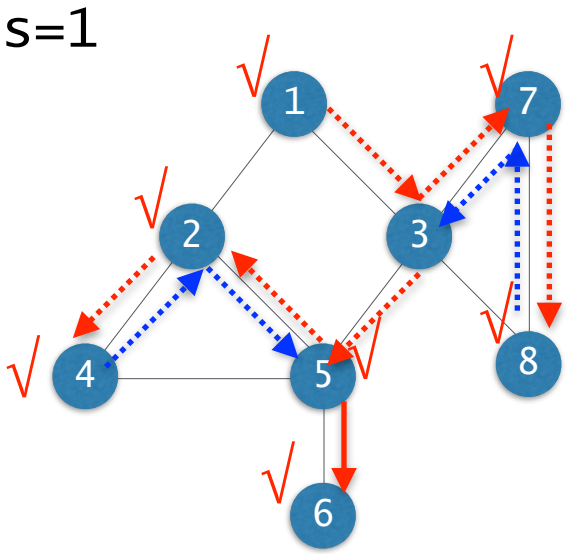
1. 출발점 s에서 시작한다.
2. 현재 노드를 visited로 mark하고 인접한 노드들 중 unvisited 노드가 존재하면 그 노드로 간다.
3. 2번을 계속 반복한다.
4. unvisited인 이웃 노드가 존재하지 않는 동안 계속해서 직전 노드로 되돌아간다.
5. 다시 2번을 반복한다.

깊이우선순회 (DFS)



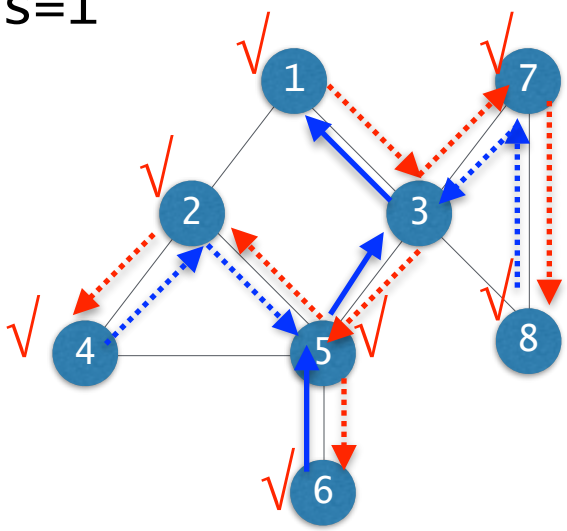
1. 출발점 s에서 시작한다.
2. 현재 노드를 visited로 mark하고 인접한 노드들 중 unvisited 노드가 존재하면 그 노드로 간다.
3. 2번을 계속 반복한다.
4. unvisited인 이웃 노드가 존재하지 않는 동안 계속해서 직전 노드로 되돌아간다.
5. 다시 2번을 반복한다.

깊이우선순회 (DFS)



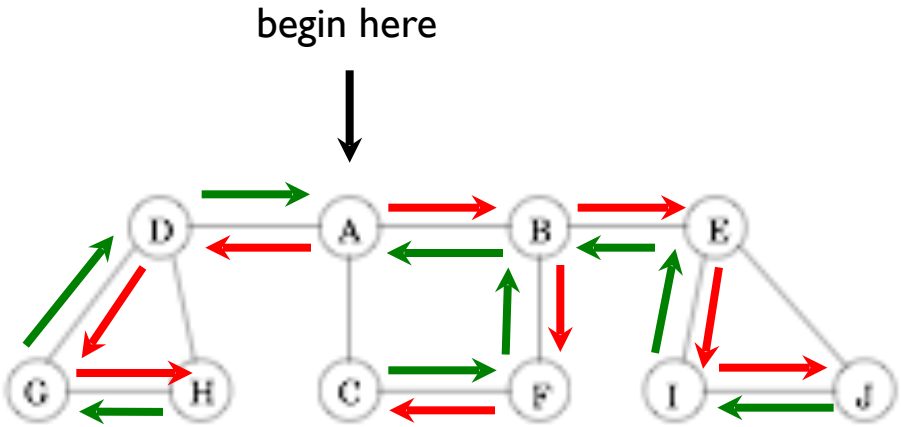
1. 출발점 s에서 시작한다.
2. 현재 노드를 visited로 mark하고 인접한 노드들 중 unvisited 노드가 존재하면 그 노드로 간다.
3. 2번을 계속 반복한다.
4. unvisited인 이웃 노드가 존재하지 않는 동안 계속해서 직전 노드로 되돌아간다.
5. 다시 2번을 반복한다.

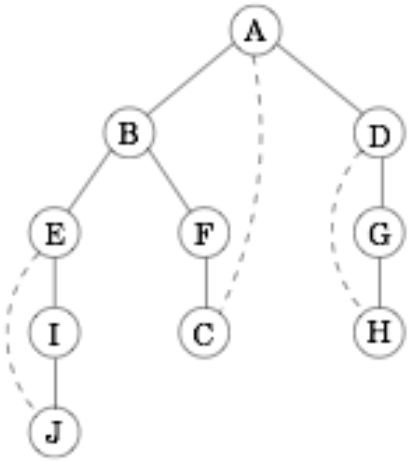
깊이우선순회 (DFS)



1. 출발점 s에서 시작한다.
2. 현재 노드를 visited로 mark하고 인접한 노드들 중 unvisited 노드가 존재하면 그 노드로 간다.
3. 2번을 계속 반복한다.
4. unvisited인 이웃 노드가 존재하지 않는 동안 계속해서 직전 노드로 되돌아간다.
5. 다시 2번을 반복한다.
6. 시작노드 s로 돌아오고 더 이상 갈 곳이 없으면 종료한다.

깊이우선순회 (DFS)





DFS 깊이우선탐색

```
DFS(G, v)
   visited[v] ← YES;
   for each node u adjacent to v do
     if visited[u] = NO then
        DFS(G, u);
   end.
end.
```

DFS 깊이우선탐색

- 그래프가 disconnected이거나 혹은 방향 그래프라면 DFS에 의해서 모든 노드가 방문되지 않을 수도 있음
- DFS를 반복하여 모든 노드 방문

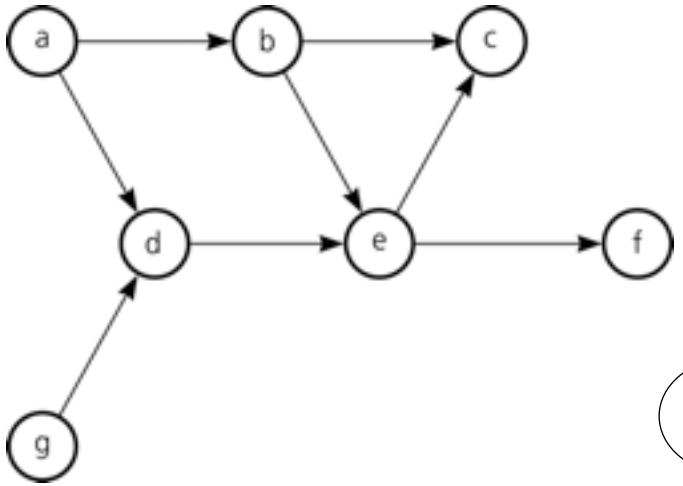
```
DFS-ALL(G)
{
    for each v ∈ V
        visited[v] ← NO;
    for each v ∈ V
        if (visited[v] = NO)  then
           DFS(G, v);
}
```

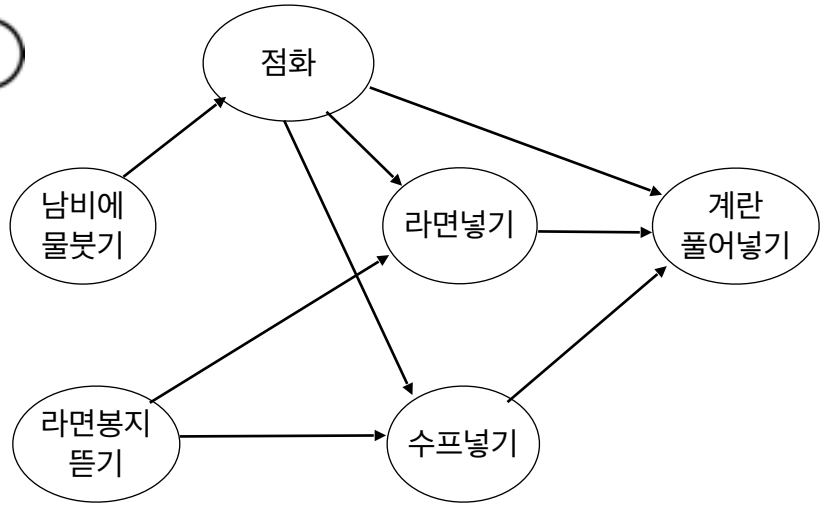
시간복잡도: O(n+m)

DAG
(Directed Acyclic Graph)

Directed Acyclic Graph

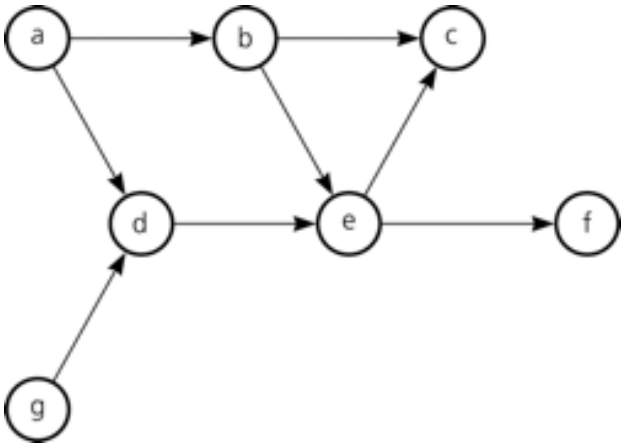
- DAG는 방향 사이클(directed cycle)이 없는 방향 그래프.
- 예: 작업들의 우선순위



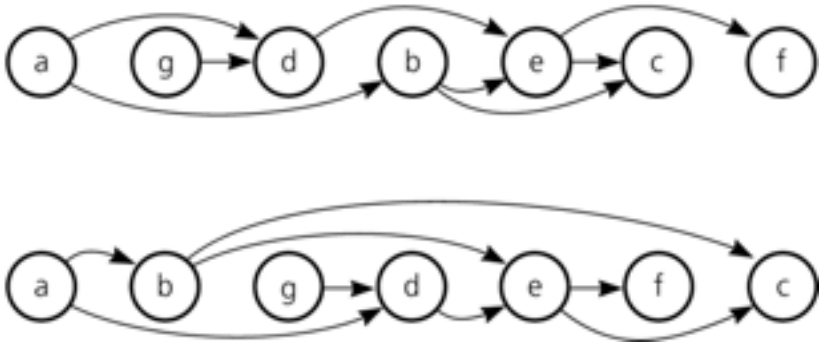


위상정렬(topological ordering)

- DAG에서 노드들의 순서화 $v_1, v_2, \ldots, v_n$, 단, 모든 에지 (v_i, v_j)에 대해서 $i<j$가 되도록.



이 그래프에 대한
위상정렬의 예 2개

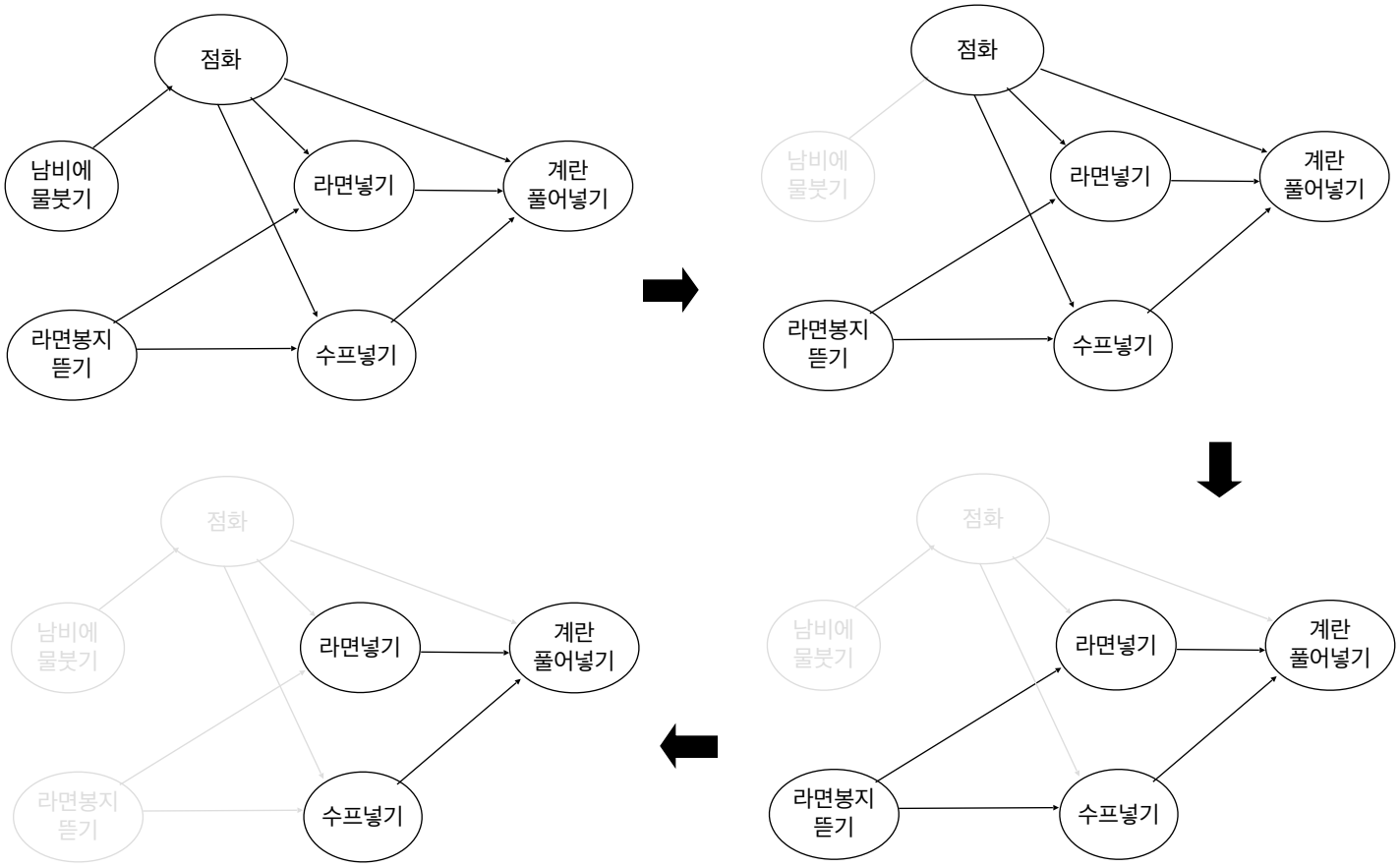


위상정렬 알고리즘 1

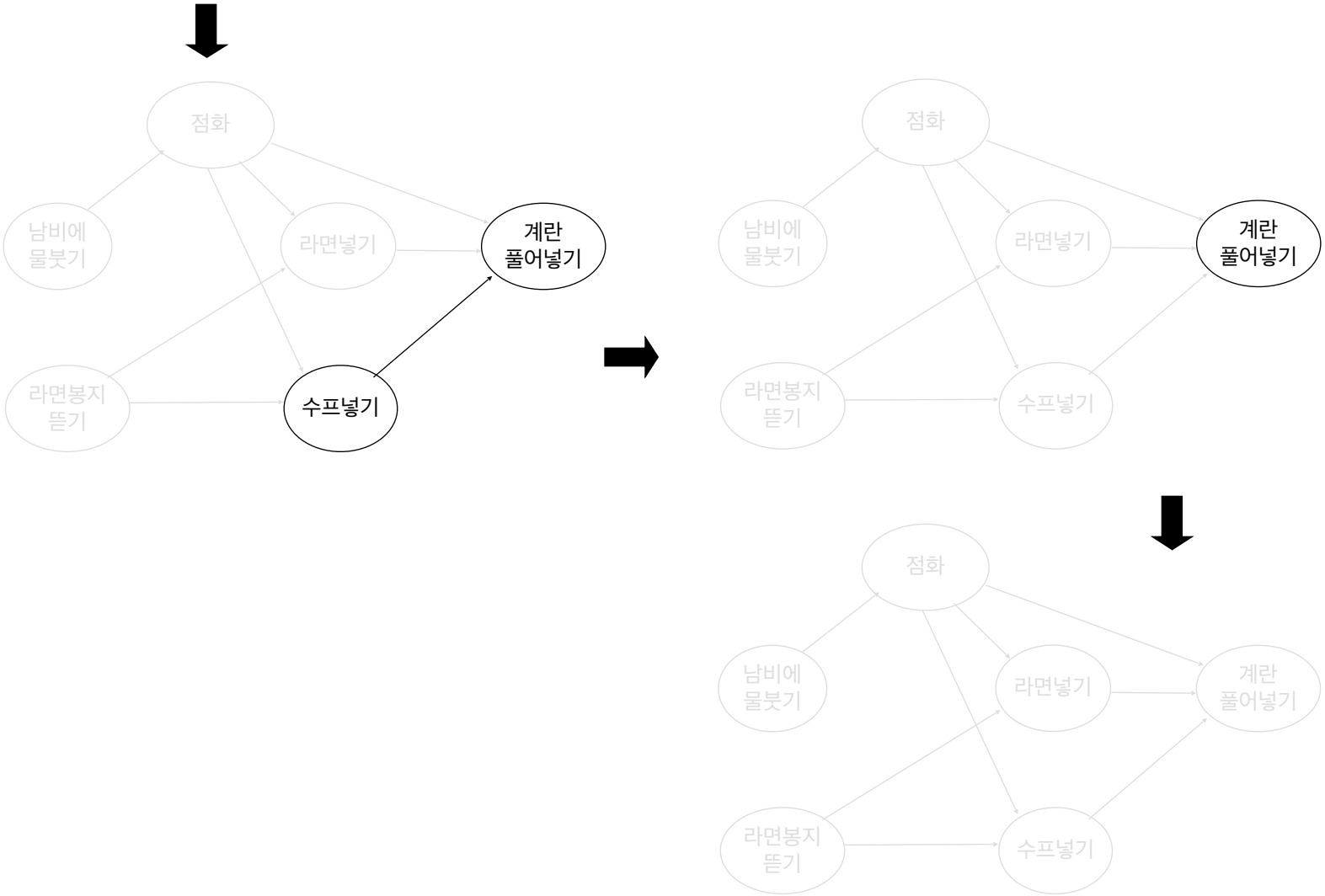
```
topologicalSort1(G)
{
    for ← 1 to n {
        진입간선이 없는 임의의 정점 u를 선택한다;
        A[i] ← u;
        정점 u와 u의 진출간선을 모두 제거한다;
    }
    ▷ 배열 A[1…n]에는 정점들을 위상정렬되어 있다
}
```

수행시간: Θ(n+m)

위상정렬 알고리즘 1의 예



위상정렬 알고리즘 1의 예 (계속)



위상정렬 알고리즘 2

```
topologicalSort2(G)
{
    for each v∈V
        visited[v] ← NO;
    make an empty linked list R;
    for each v∈V    ▷ 정점의 순서는 상관없음
        if (visited[v] = NO) then
            DFS-TS(v, R);
}
```

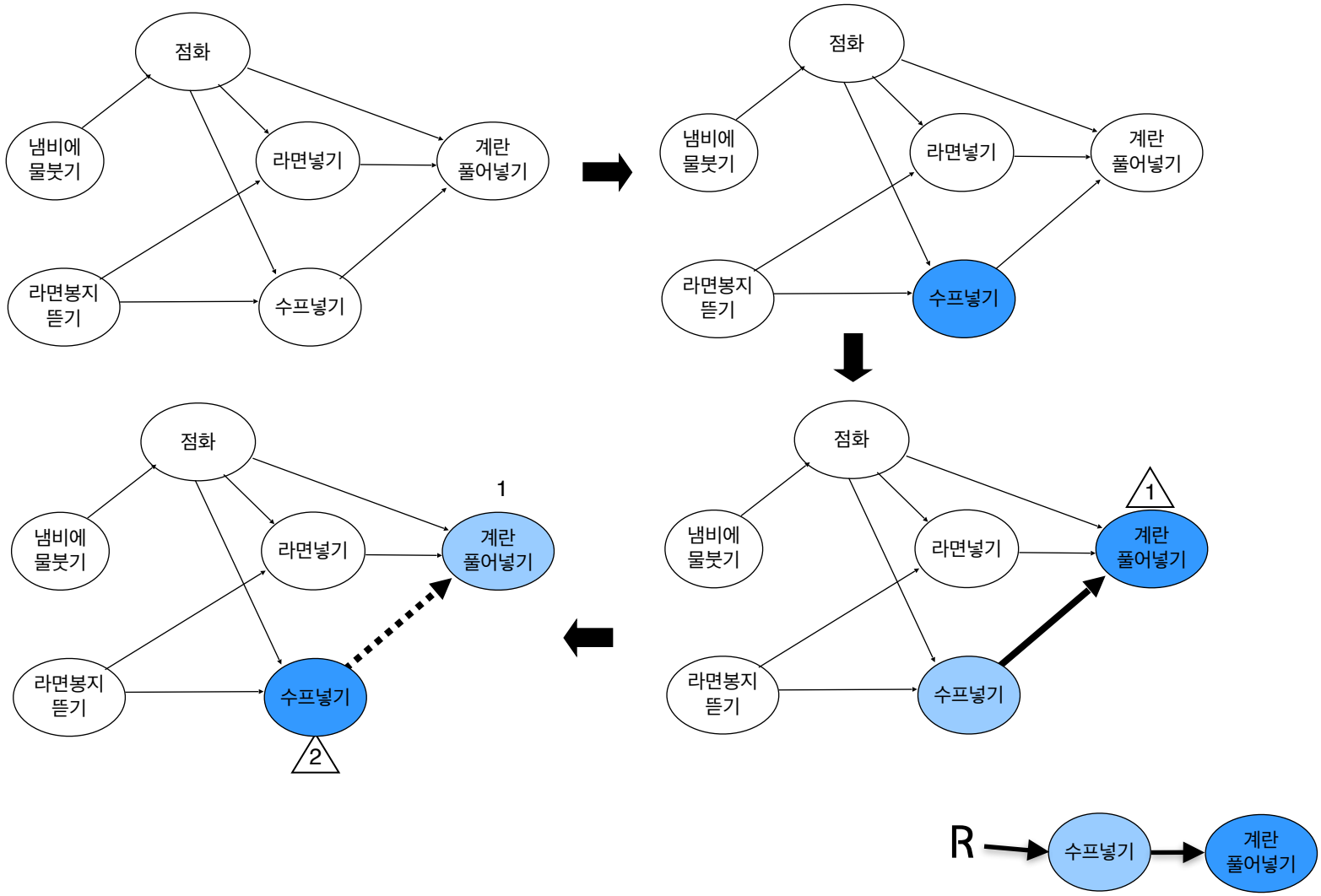
위상정렬 알고리즘 2

```
DFS-TS(v, R)
{
    visited[v] ← YES;
    for each x adjacent to v do
       if (visited[x] = NO) then
          DFS-TS(x, R) ;
    add v at the front of the linked list R;
}
```

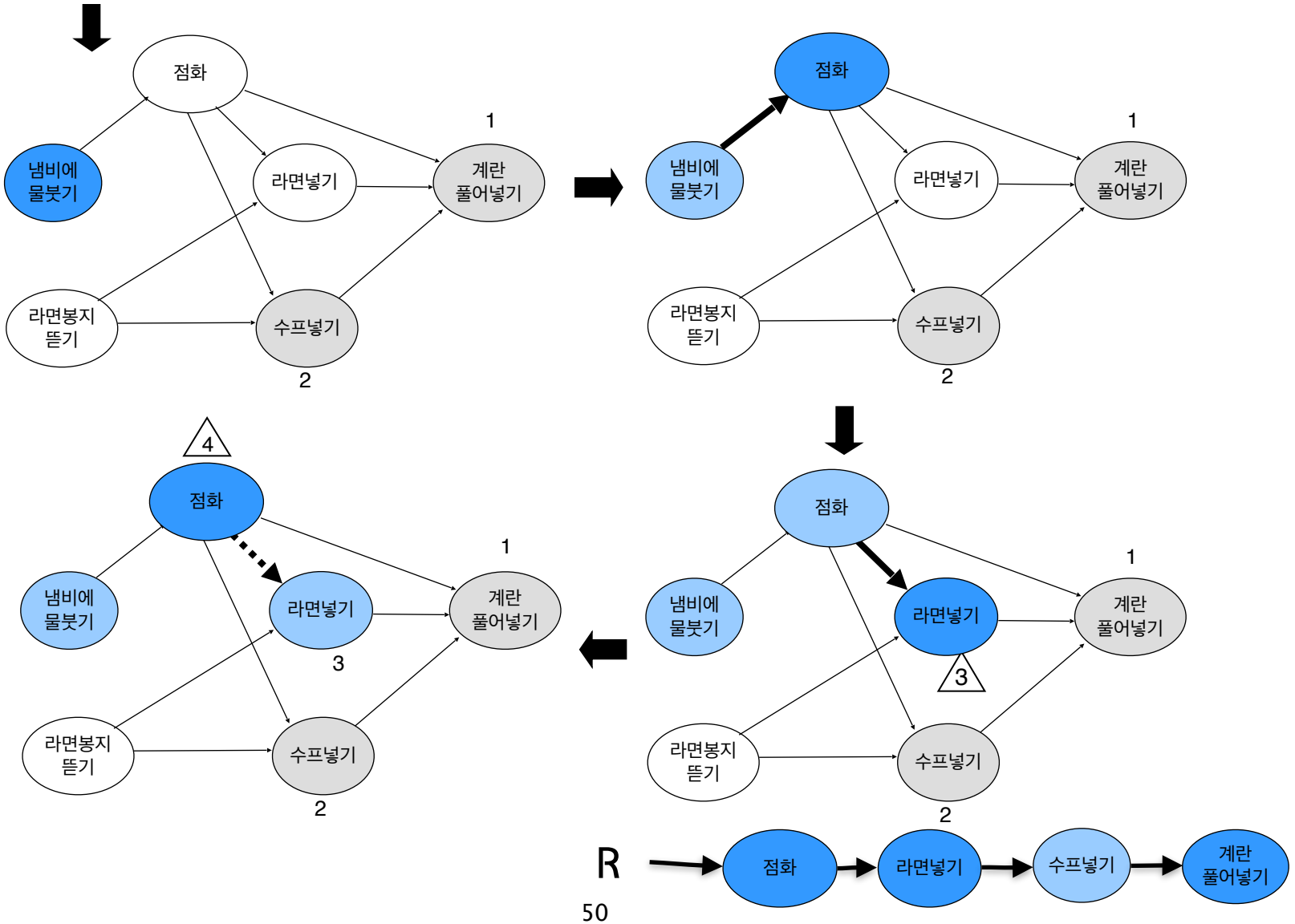
알고리즘이 끝나면 연결 리스트 R에는 정점들이 위상정렬된 순서로 매달려 있다.

수행시간: $\Theta(n+m)$

위상정렬 알고리즘 2의 작동 예



위상정렬 알고리즘 2의 작동 예 (계속)



위상정렬 알고리즘 2의 작동 예 (계속)