COSE213: Data Structure

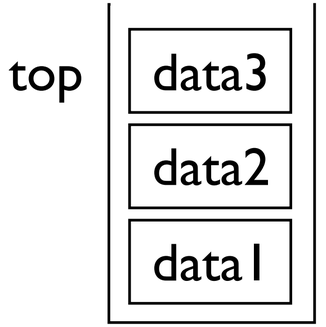
Lecture 12 - 그래프 (Graph)

Minseok Jeon

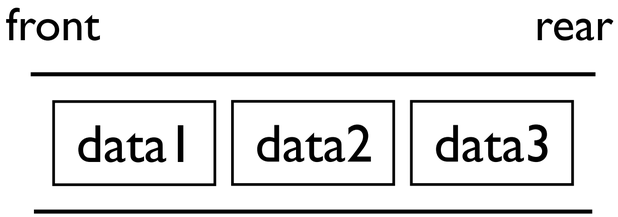
2024 Fall

리뷰: 선형 (linear) 자료구조

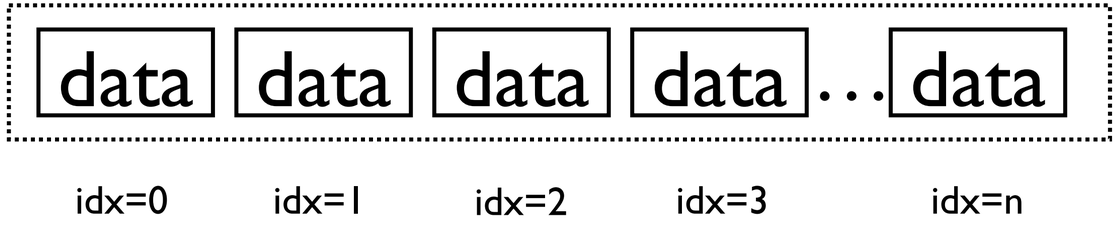
- 스택(Stack) 자료구조: 후입선출(LIFO) 원칙을 따르는 자료구조



- 큐(Queue) 자료구조: 선입선출 원칙을 따르는 자료구조

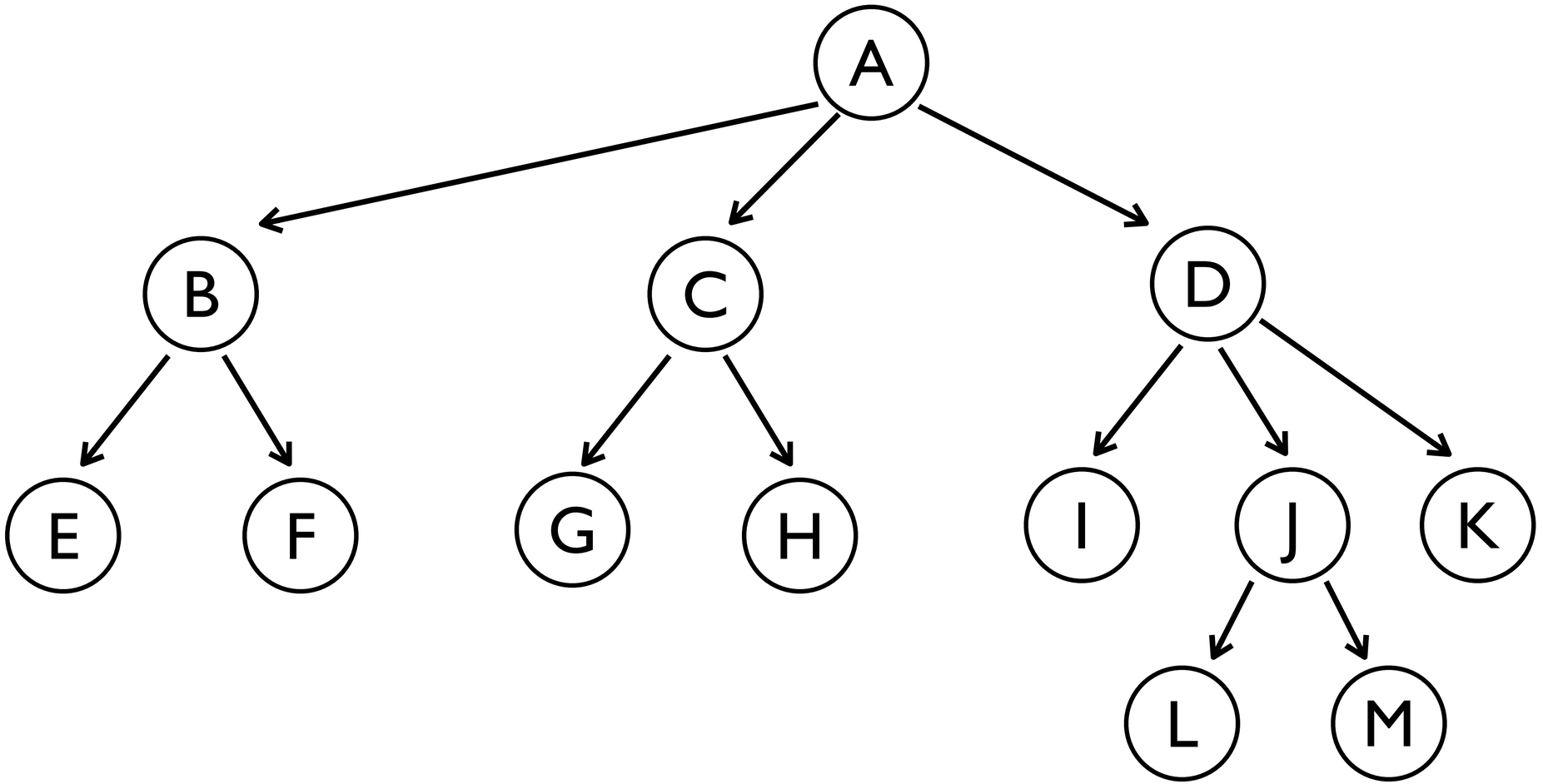


- 리스트(list) 자료구조: 데이터들이 **순차적**으로 나열되어 있는 선형 자료구조



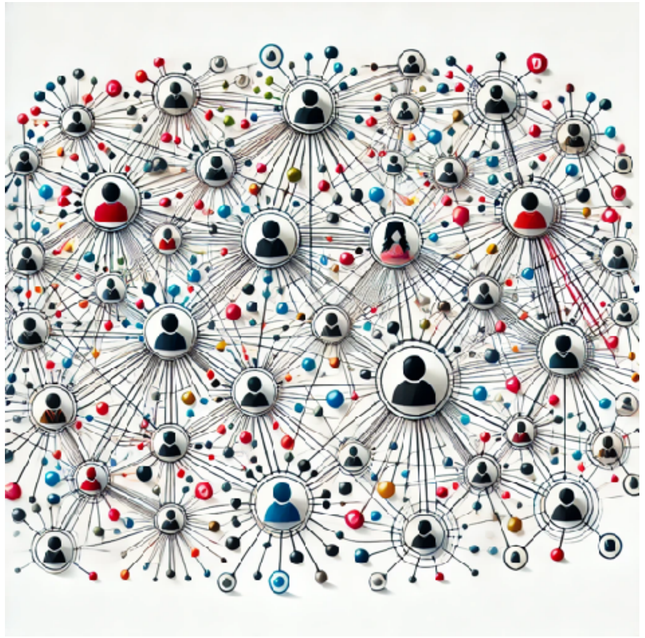
리뷰: 트리 (Tree)

- 트리는 계층적 자료를 표현하는데 이용하는 비 선형 (non-linear) 자료구조

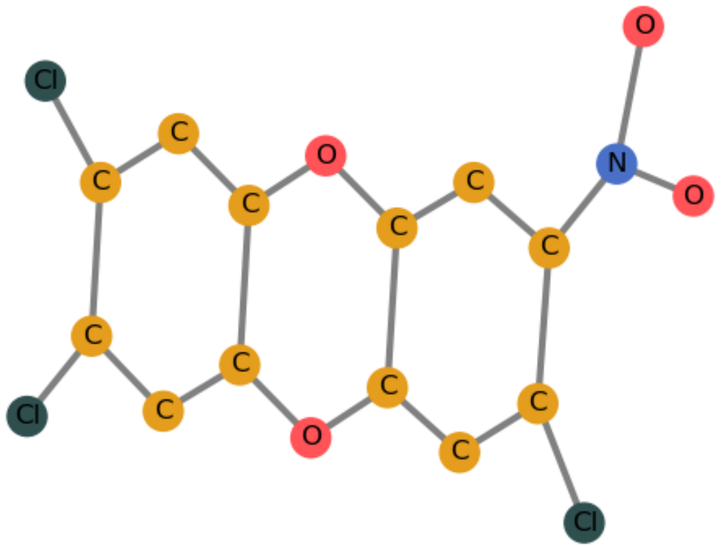


문제: 데이터간의 관계를 표현해야 하는 경우

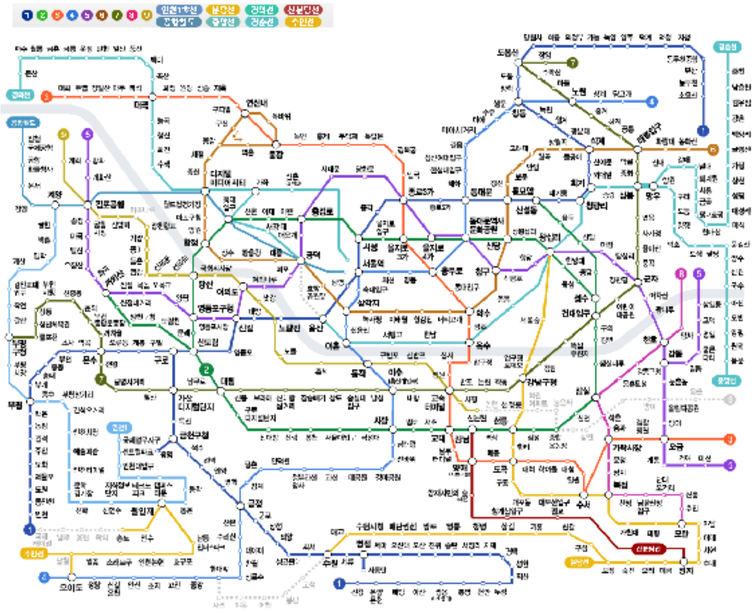
- 현실의 많은 데이터들은 데이터간의 관계를 가지고 있음
- 데이터간 관계를 표현하기 위한 자료구조가 필요함



소셜 네트워크



분자구조

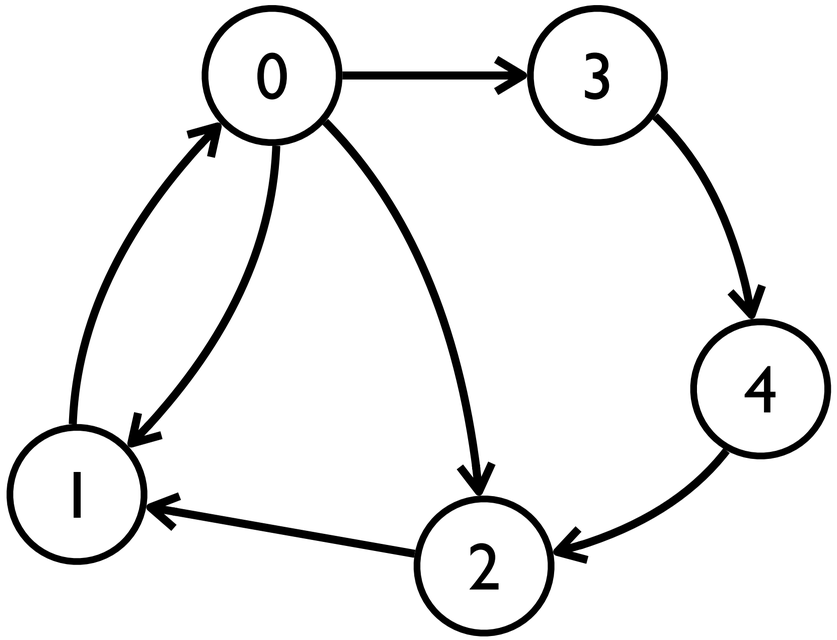


지하철 노선도

...

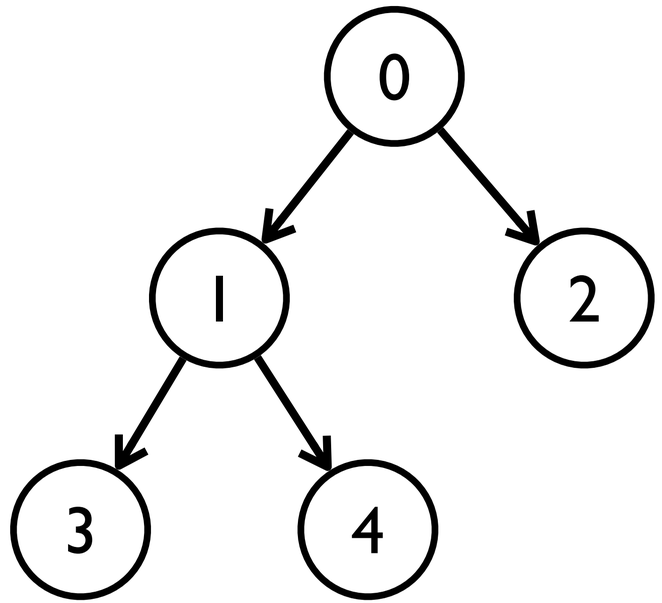
그래프 (Graph)

- 그래프는 데이터간의 (임의의) 관계를 표현할수 있는 자료구조



그래프 자료구조

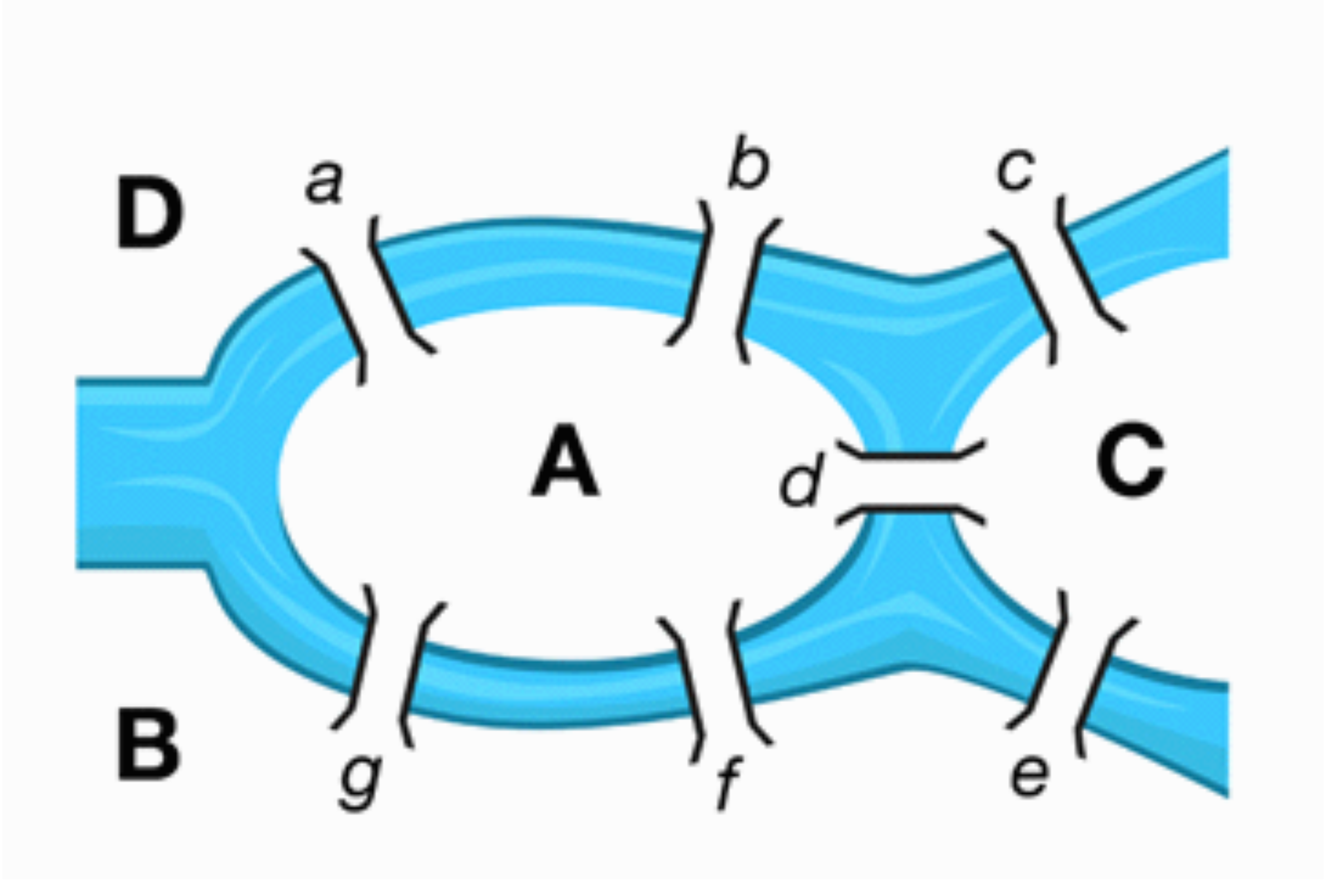
vs



트리 자료구조

쾨니히스베르크 다리 건너리 문제

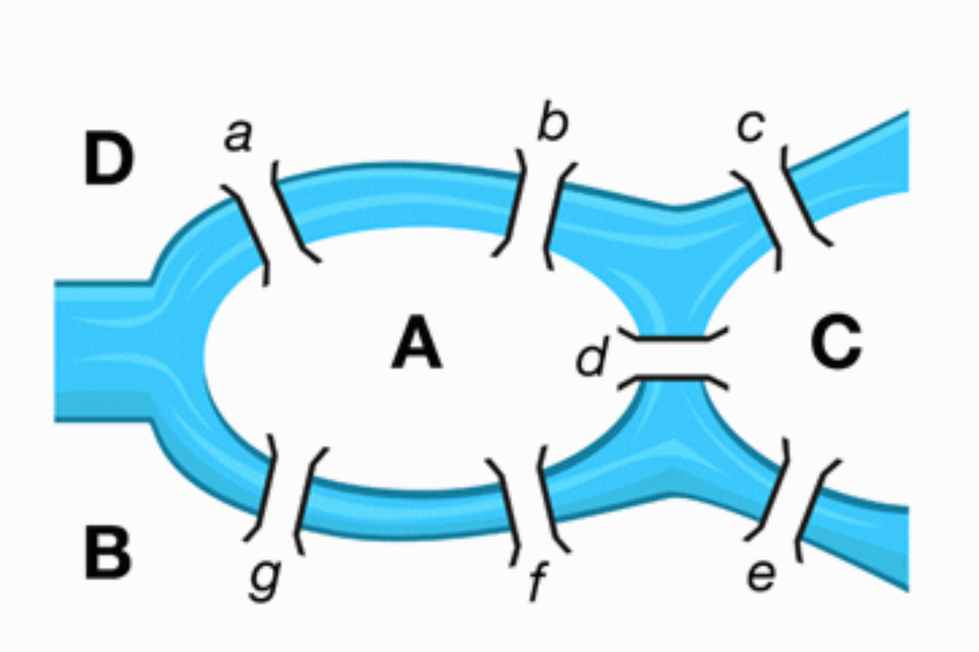
- 문제: 모든 다리를 한번만 건너서 출발했던 장소로 돌아올 수 있는가?



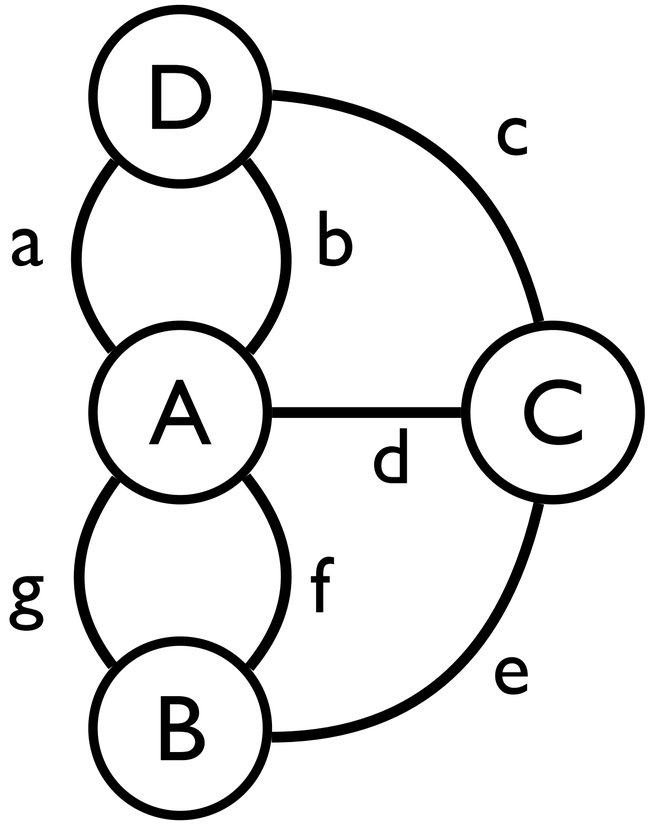
https://blog.naver.com/ebssmd02/221100136140

쾨니히스베르크 다리 건너리 문제

- 문제: 모든 다리를 한번만 건너서 출발했던 장소로 돌아올 수 있는가?

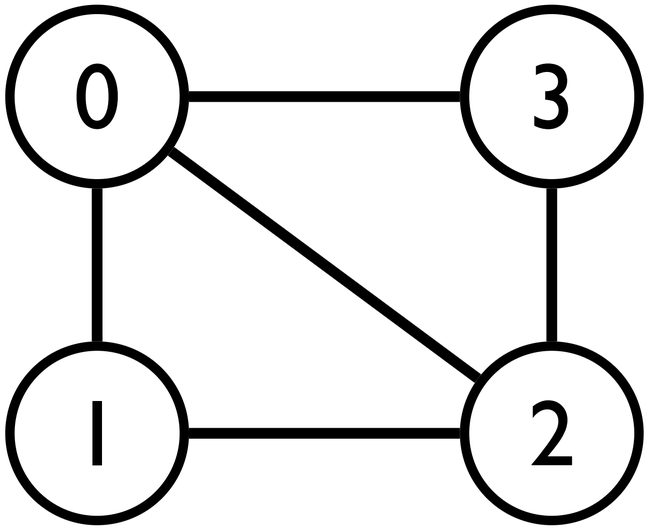


그래프로 변환



그래프 자료구조

- 그래프 자료구조는 노드(vertex)와 간선(edge)들의 집합임



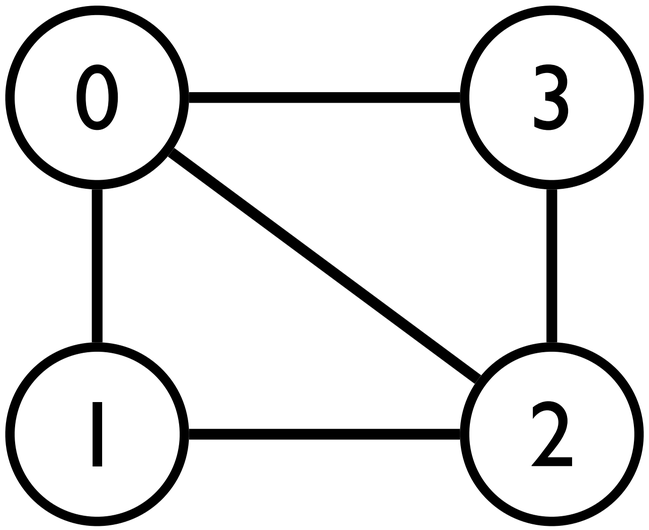
그래프 G = (V, E)

노드 (V) = {0, 1, 2, 3}

간선 (E) = {(0, 1), (0, 2), (0, 3), (1, 2), (2, 3)}

그래프의 종류

- 무방향 그래프 (undirected graph): 간선에 방향이 표시되지 않은 그래프
  - 각 간선은 양방향으로 갈 수 있음을 의미함 ((A, B)는 (B, A)와 동일한 간선임)



무방향 그래프 G = (V, E)

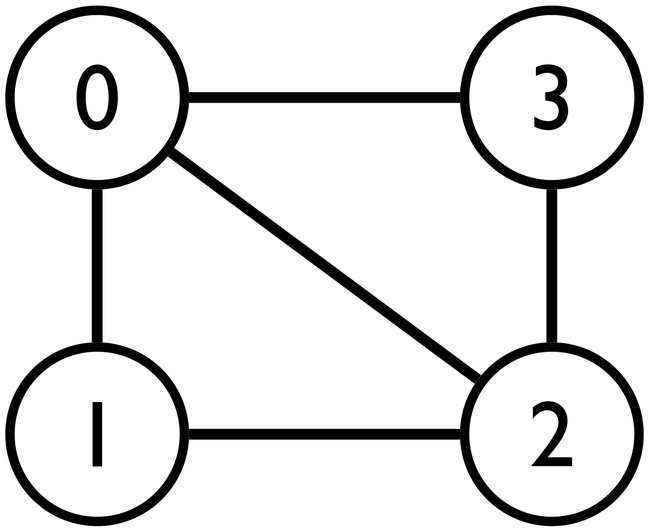
노드 (V) = {0, 1, 2, 3}

간선 (E) = {(0, 1), (0, 2), (0, 3), (1, 2), (2, 3)}

실제 사용 예시: 지하철 노선도

그래프의 종류

- 무방향 그래프 (undirected graph): 간선에 방향이 표시되지 않은 그래프
  - 각 간선은 양방향으로 갈 수 있음을 의미함 ((A, B)는 (B, A)와 동일한 간선임)

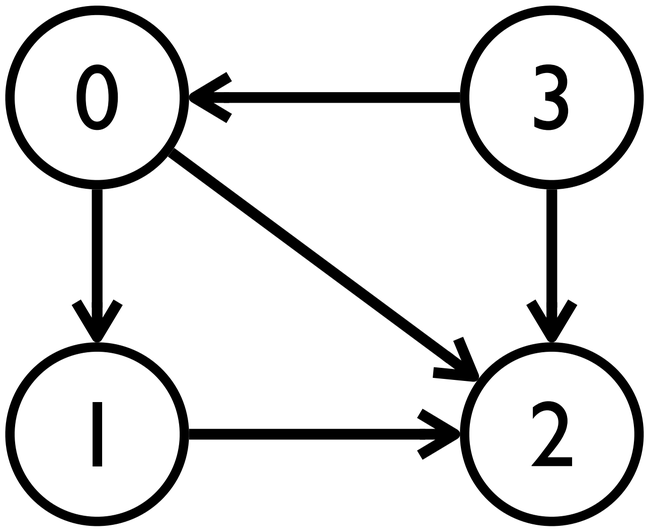


무방향 그래프 G = (V, E)

실제 사용 예시: 지하철 노선도

그래프의 종류

- 방향 그래프 (directed graph): 간선에 방향이 존재하는 그래프
  - 간선에서 한쪽 방향으로만 갈 수 있음



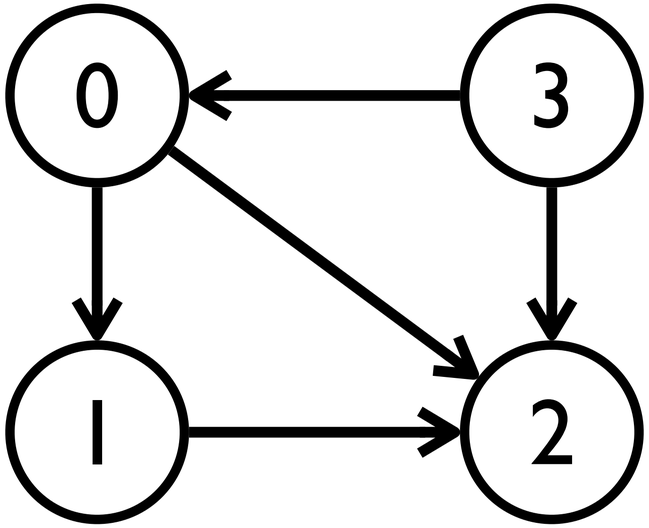
방향 그래프 G = (V, E)

노드 (V) = {0, 1, 2, 3}

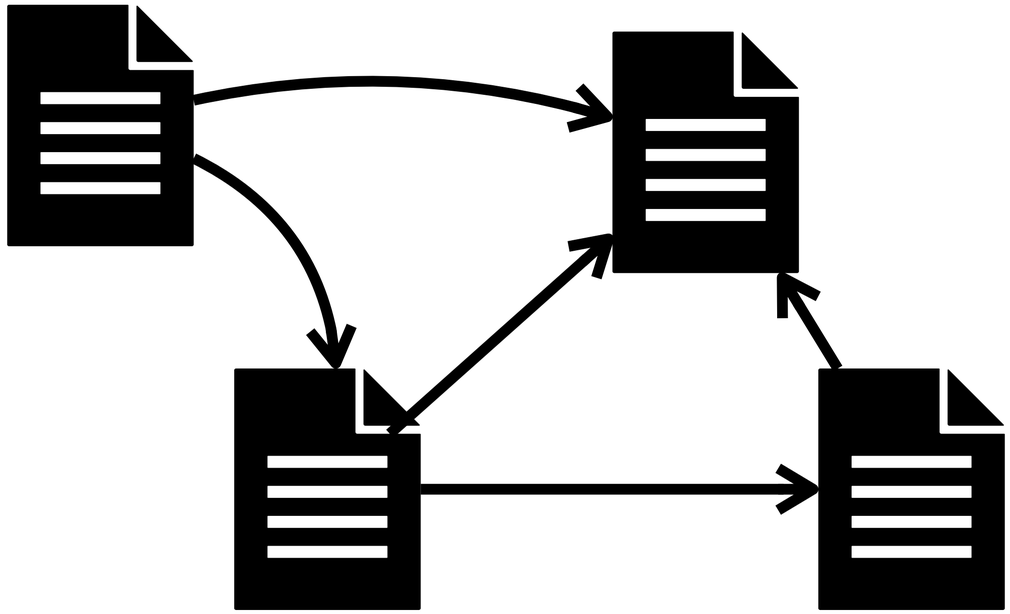
간선 (E) = {(0, 1), (0, 2), (1, 2), (3, 0), (3, 2)}

그래프의 종류

- 방향 그래프 (directed graph): 간선에 방향이 존재하는 그래프
  - 간선에서 한쪽 방향으로만 갈 수 있음



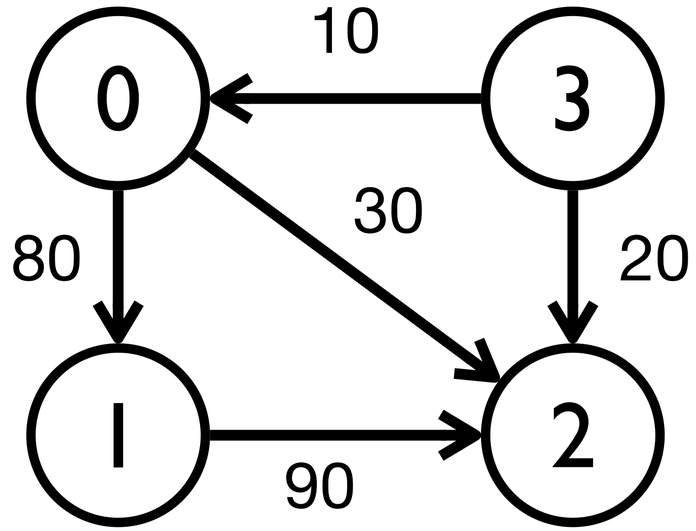
방향 그래프 G = (V, E)



실제 사용 예시: 논문 인용 데이터

그래프의 종류

- 가중치 그래프 (directed graph): 간선이 가중치를 가지고 있는 그래프
  - 간선에서 한쪽 방향으로만 갈 수 있음



가중치 그래프 G = (V, E)

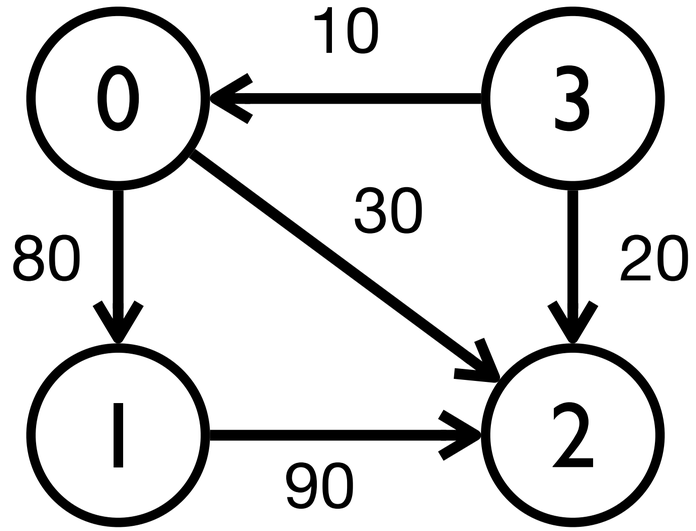
노드 (V) = {0, 1, 2, 3}

간선 (E) = {(0, 1), (0, 2), (1, 2), (3, 0), (3, 2)}

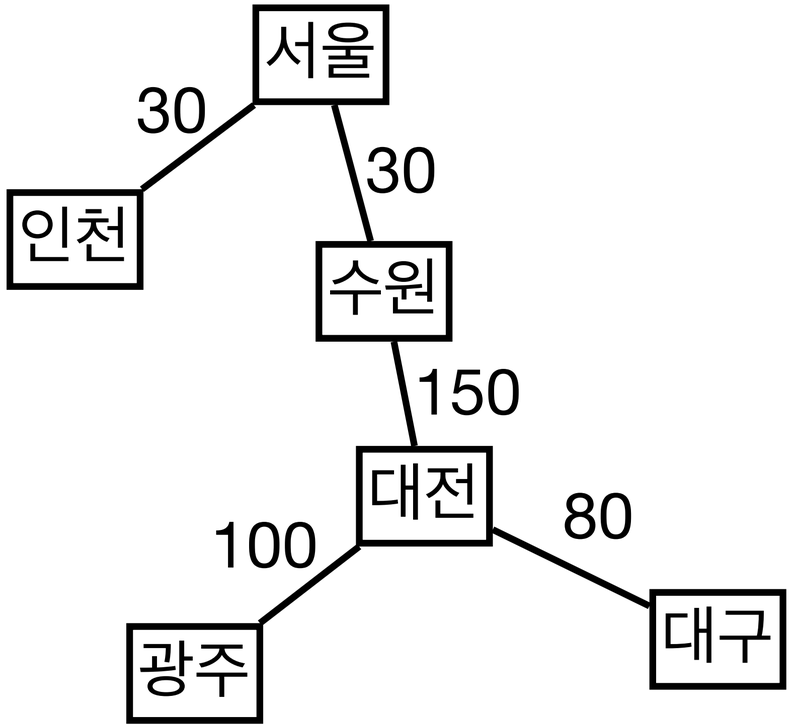
80 30 90 10 20

그래프의 종류

- 가중치 그래프 (weighted graph): 간선이 가중치를 가지고 있는 그래프
  - 간선에서 한쪽 방향으로만 갈 수 있음



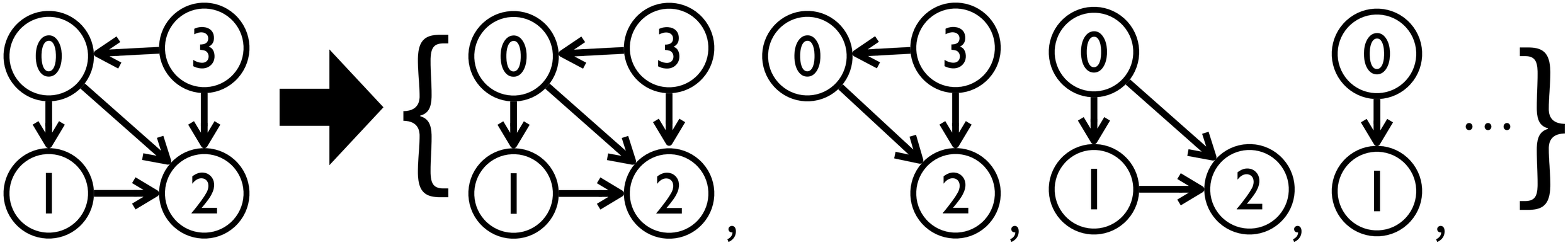
가중치 그래프 G = (V, E)



실제 사용 예시: 도시간 도로의 길이

부분 그래프(subgraph)

- 부분그래프: 그래프 G를 구성하는 노드와 간선의 집합의 부분집합으로 이루어지는 그래프

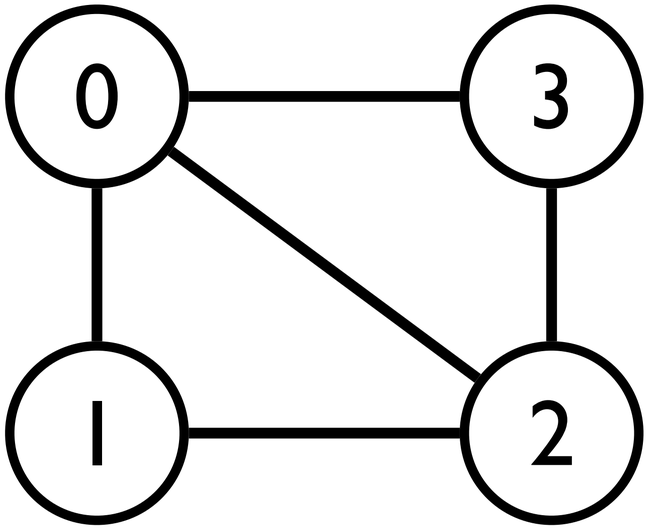


그래프 G

G의 부분 그래프 (subgraphs of G)

그래프의 용어

- 인접 노드 (adjacent vertex): 간선에 의해 직접 연결된 노드들



노드 0의 인접 노드 : {3, 1, 2}

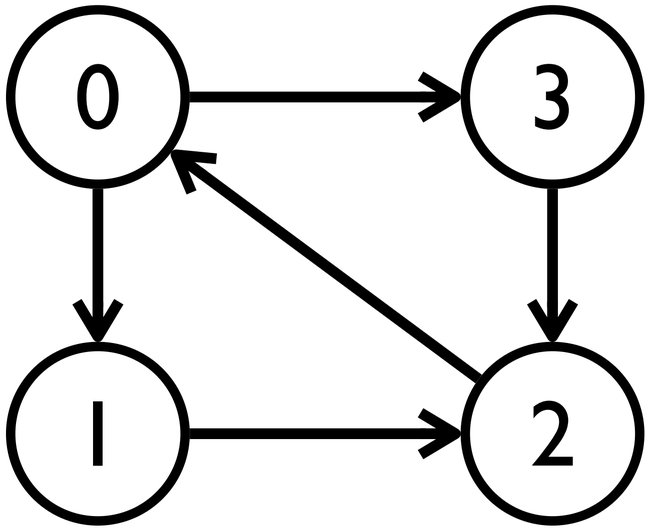
노드 1의 인접 노드 : {0, 2}

노드 2의 인접 노드 : {0, 1, 3}

노드 3의 인접 노드 : {0, 2}

그래프의 용어

- 노드의 차수 (degree): 노드에 연결된 간선(edge)의 수
  - 진출 차수(out-degree): 노드에서 나가는 간선의 수
  - 진입 차수(in-degree): 노드로 진입하는 간선의 수



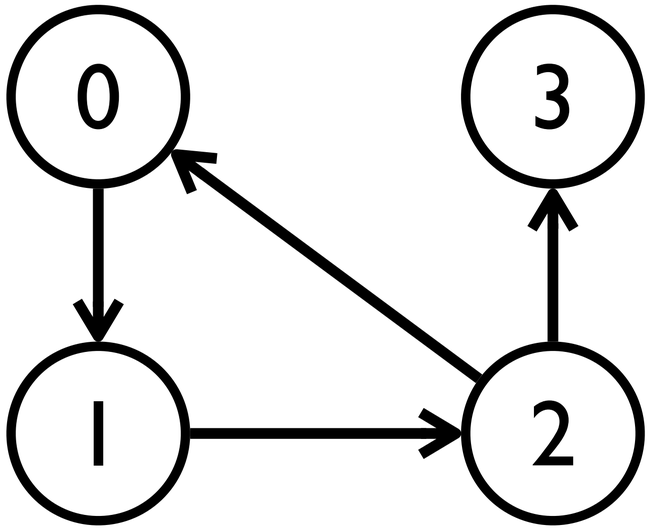
노드 0의 차수 : 3

노드 0의 진입 차수: 1

노드 0의 진출 차수: 2

그래프의 용어

- 경로(path): 출발 노드부터 도착 노드까지 간선을 따라 갈 수 있는 길.
  - 단순 경로: 경로 중에서 반복되는 간선이 없는 경로
- 경로의 길이: 경로를 구성하는데 사용된 간선의 수

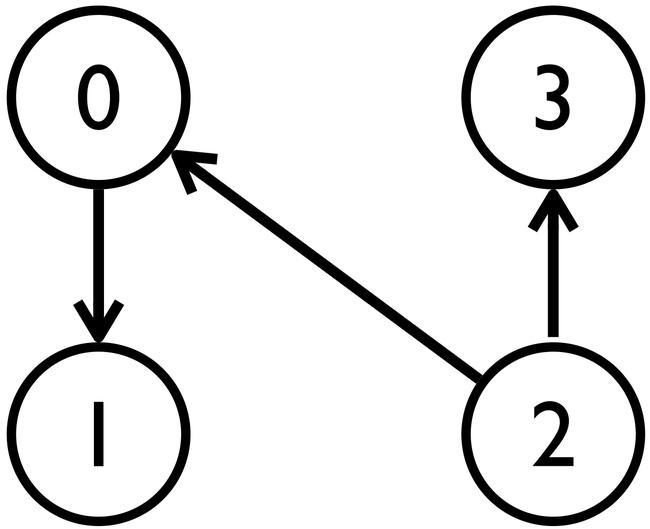


- 노드 0에서 노드3으로의 단순 경로:

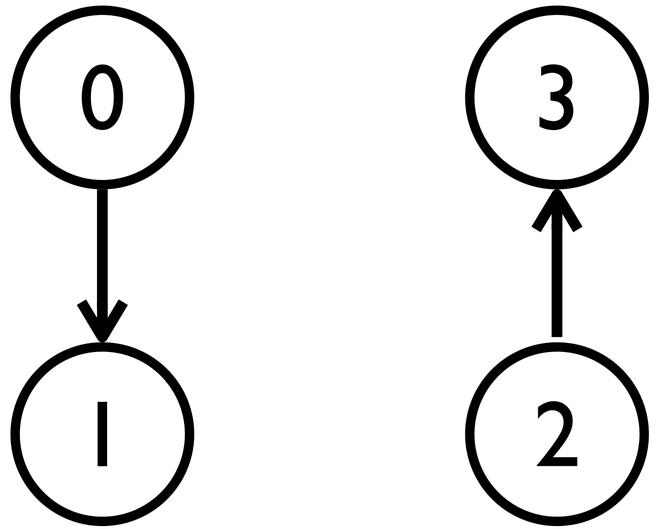
  0, 1, 2, 3

그래프의 용어

- 연결 그래프(connected graph): 그래프의 모든 노드들 사이에 경로가 존재함
- 단절 그래프(disconnected graph): 그래프의 모든 노드들 사이에 경로가 존재하지 않는 경우가 있음



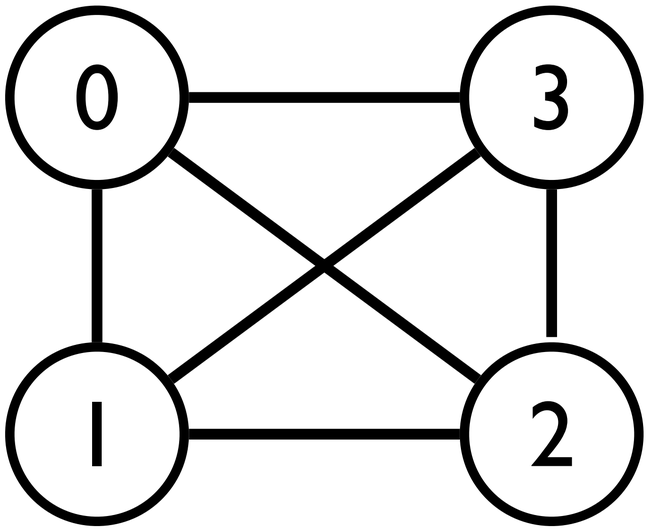
연결 그래프



단절 그래프

그래프의 용어

- 완전 그래프(complete graph): 모든 노드간에 간선이 존재하는 그래프
  - 무방향 그래프에서 노드의 개수가 n일 때 간선의 수는 n*(n-1)/2개임



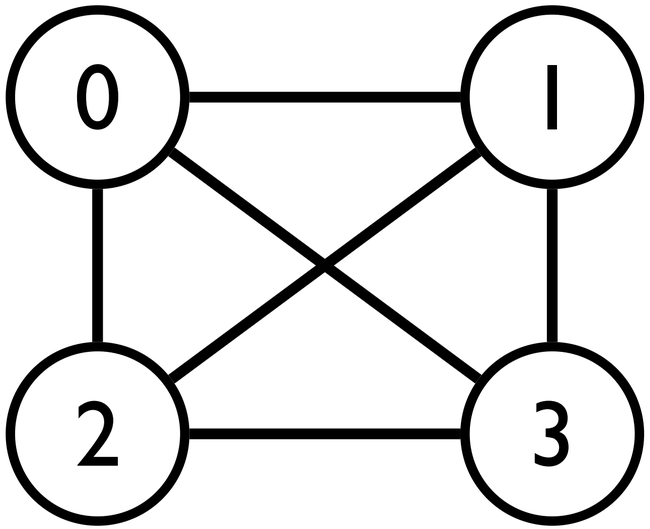
완전 그래프

노드의 개수 : 4

간선의 개수 : 6

그래프의 표현방법 1: 인접 행렬 (Adjacency Matrix)

- 노드의 개수가 n일때 n×n의 2차원 배열의 형태인 인접 행렬로 그래프를 표현할 수 있음
- 인접 행렬은 2차원 배열로 표현함



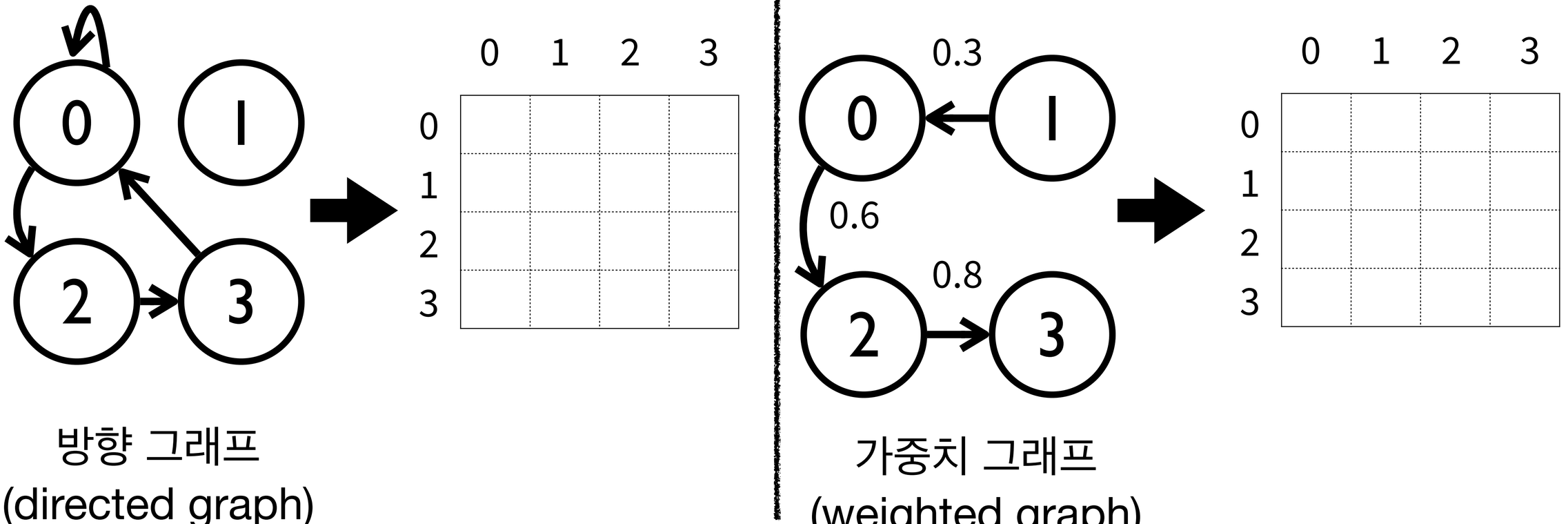
무방향 그래프
(undirected graph)

	0	1	2	3
0	0	1	1	1
1	1	0	1	1
2	1	1	0	1
3	1	1	1	0

인접 행렬 M

그래프의 표현방법 1: 인접 행렬 (Adjacency Matrix)

- 노드의 개수가 n일때 n×n의 2차원 배열의 형태인 인접 행렬로 그래프를 표현할 수 있음

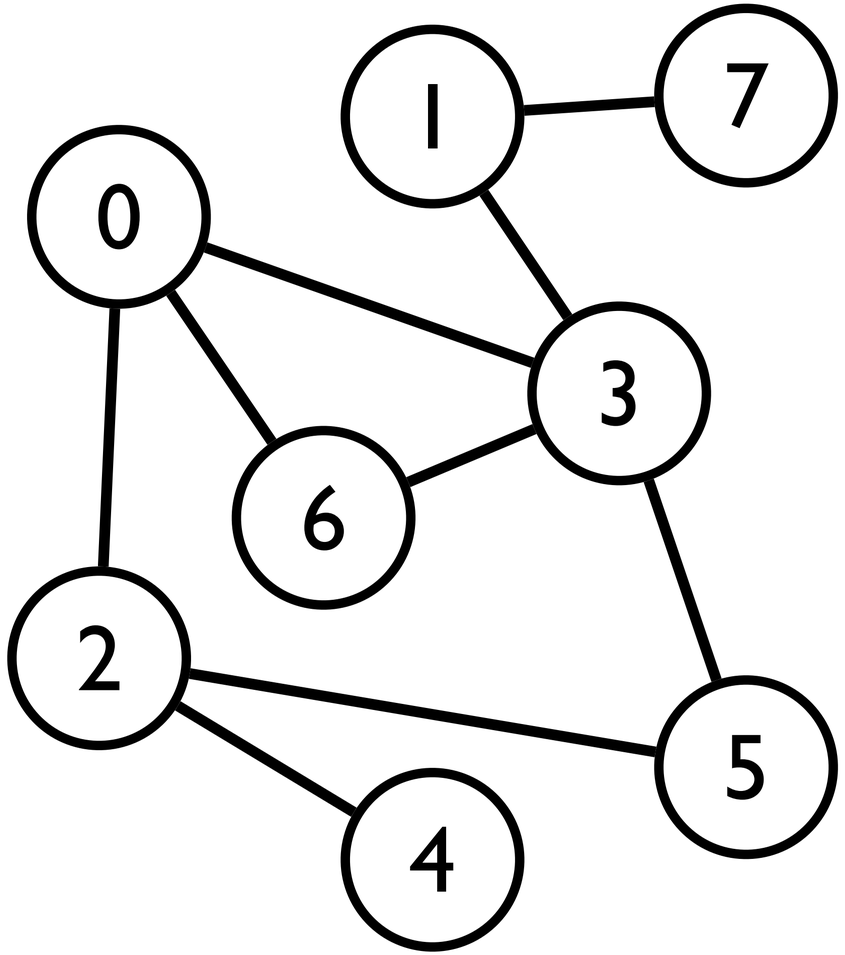


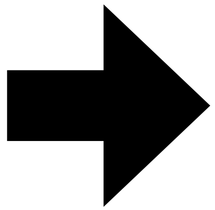
방향 그래프
(directed graph)

가중치 그래프
(weighted graph)

그래프의 표현방법 1: 인접 행렬 (Adjacency Matrix)

- 노드의 개수가 n일때 n×n의 2차원 배열의 형태인 인접 행렬로 그래프를 표현할 수 있음

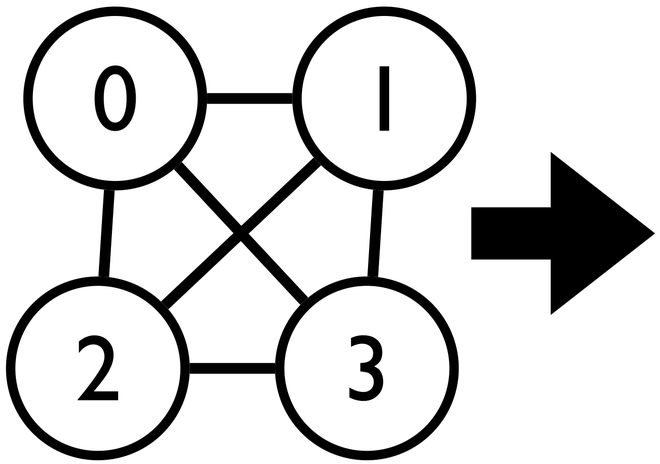




	0	1	2	3	4	5	6	7
0								
1								
2								
3								
4								
5								
6								
7								

그래프의 표현방법 1: 인접 행렬 (Adjacency Matrix)

- 그래프를 인접행렬로 표현했을 때의 장점

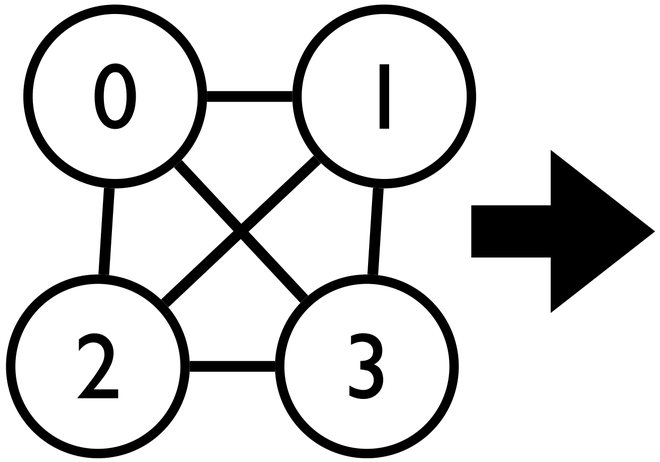


0	1	1	1
1	0	1	1
1	1	0	1
1	1	1	0

인접 행렬 M

그래프의 표현방법 1: 인접 행렬 (Adjacency Matrix)

- 그래프를 인접행렬로 표현했을 때의 장점
  - 두 노드 사이에 간선이 존재하는지 여부를 $O(1)$ 시간 안에 즉시 알아낼 수 있음



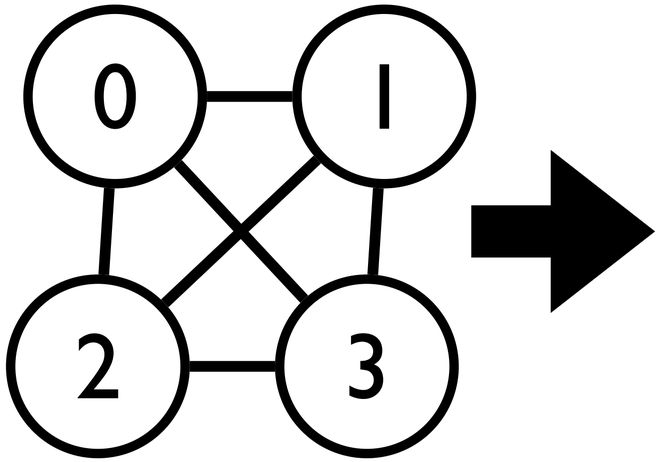
0	1	1	1
1	0	1	1
1	1	0	1
1	1	1	0

인접 행렬 M

```
procedure edgeExist(i, j)
  if M[i][j] = 1 then
    return true
  else:
    return false
end procedure
```

그래프의 표현방법 1: 인접 행렬 (Adjacency Matrix)

- 그래프를 인접행렬로 표현했을 때의 장점
  - 두 노드 사이에 간선이 존재하는지 여부를 $O(1)$ 시간 안에 즉시 알아낼 수 있음
  - 노드의 차수를 인접행렬의 행이나 열을 조사하면 $O(n)$안에 알 수 있음



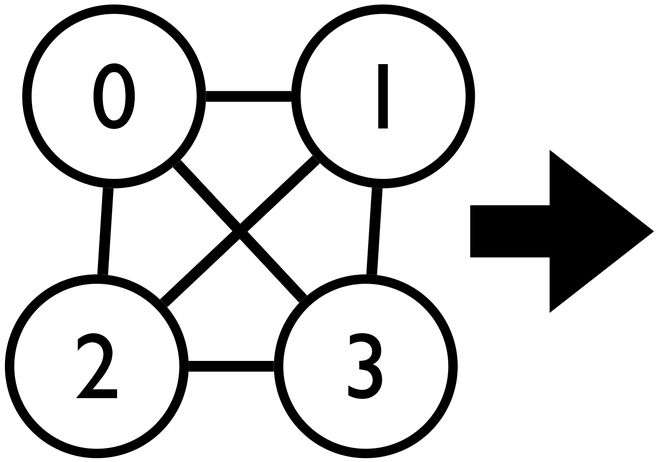
0	1	1	1
1	0	1	1
1	1	0	1
1	1	1	0

인접 행렬 M

```
procedure outDegree(M, i)
  degree ← 0
  for k = 0 to n-1 do
    degree  ← M[i][k]
  end for
  return degree
end procedure
```

그래프의 표현방법 1: 인접 행렬 (Adjacency Matrix)

- 그래프를 인접행렬로 표현했을 때의 단점
  - n개의 노드를 가지는 그래프를 행렬로 표시하기 위해서는 간선의 수와 무관하게 항상 n^2개의 공간이 필요함
  - 총 간선의 수를 알아내려면 n^2번의 조사가 필요함 ($O(n^2)$)

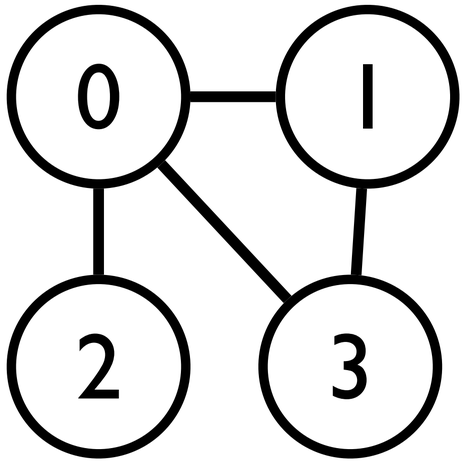


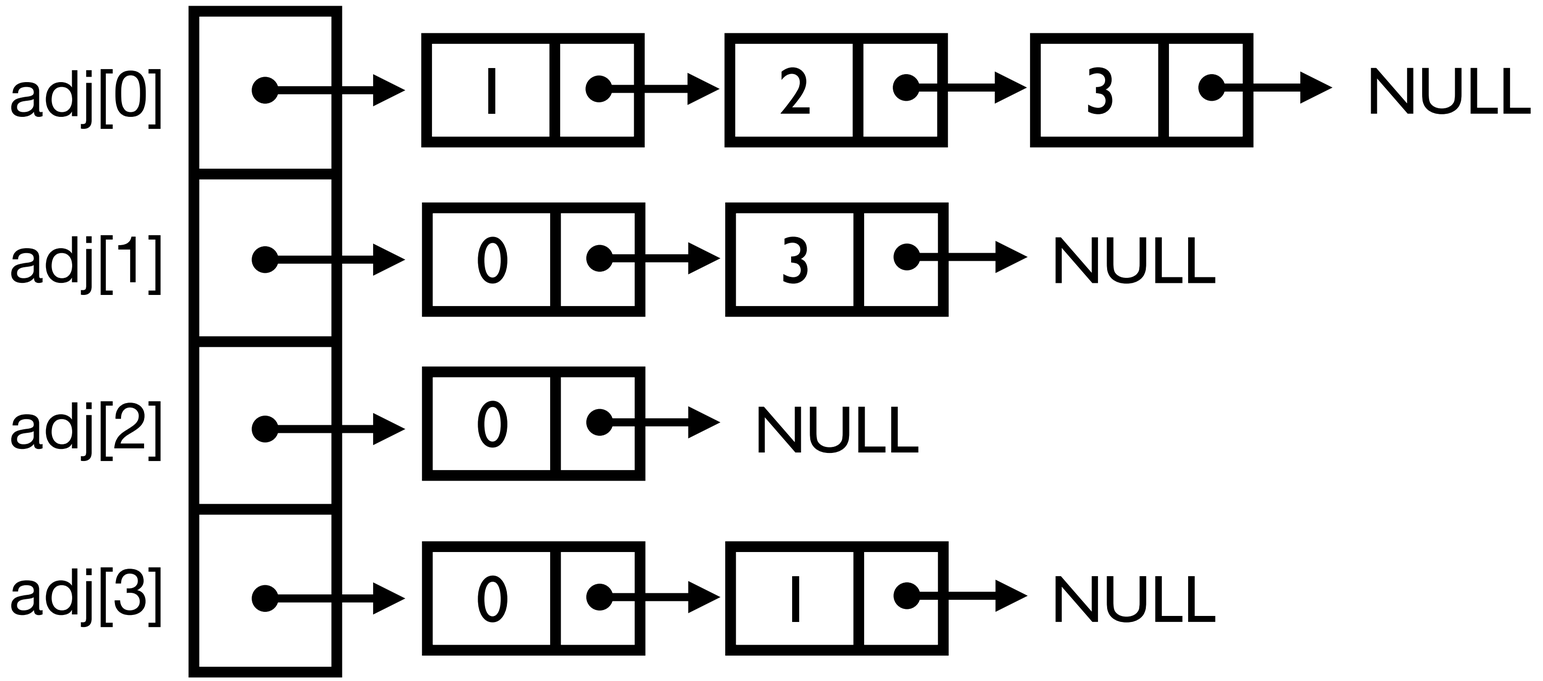
0	1	1	1
1	0	1	1
1	1	0	1
1	1	1	0

인접 행렬 M

그래프의 표현방법 2: 인접 리스트 (Adjacency List)

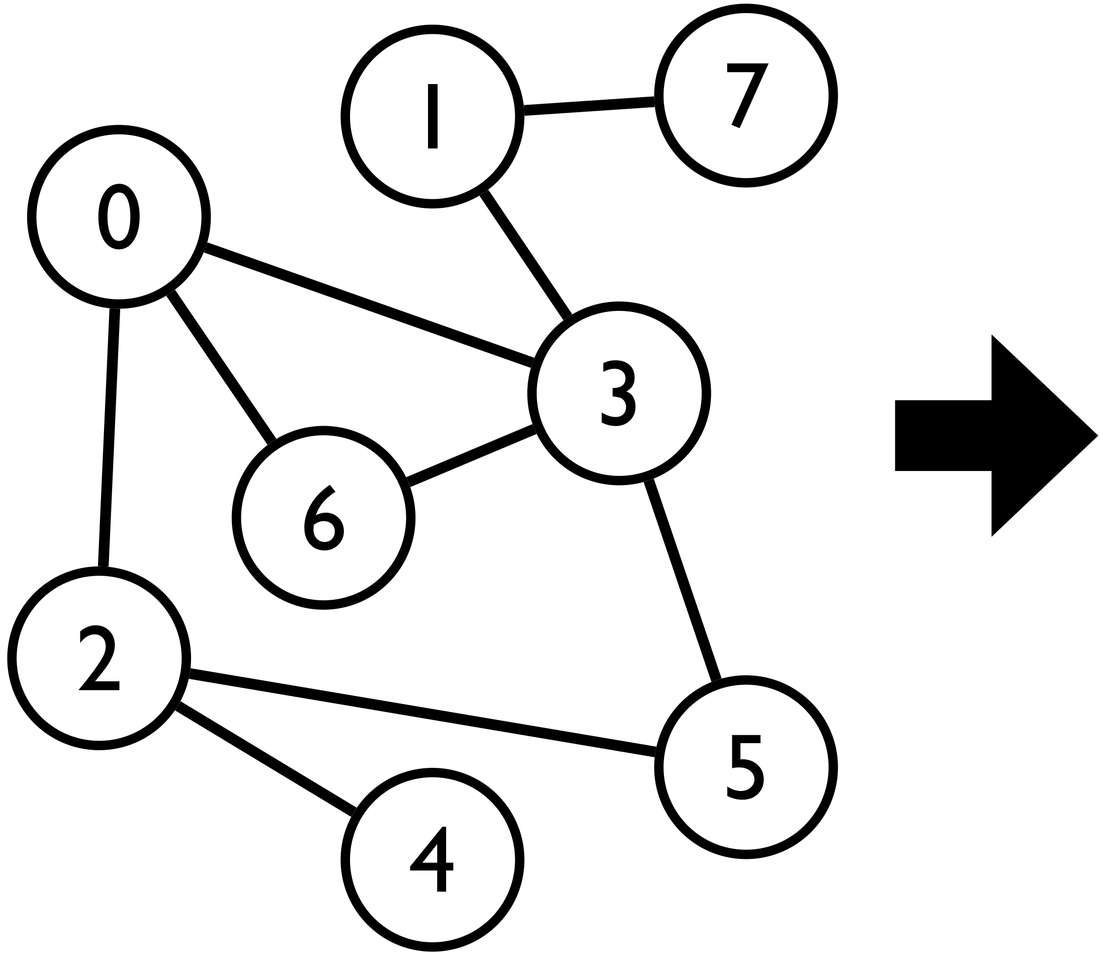
- 그래프의 각 노드에 인접한 노드들을 연결 리스트로 표현할 수 있음.





그래프의 표현방법 2: 인접 리스트 (Adjacency List)

- 그래프의 각 노드에 인접한 노드들을 연결 리스트로 표현할 수 있음.



그래프 데이터

- 그래프 데이터를 관리하기 위해 다음의 기능들을 제공해야함
    - insert node: 그래프 데이터에 새로운 노드를 삽입함
    - insert edge: 그래프 데이터에 새로운 간선을 추가함
    - delete node: 그래프 데이터에 있는 노드를 삭제함
    - delete edge: 그래프 데이터에 있는 간선을 삭제함
    - search: 그래프 데이터에서 주어진 키를 가진 노드를 반환함

인접 행렬을 사용한 그래프 구현 예시

- 그래프 자료구조는 다음과 같은 정보를 가지는 데이터 타입

```
#define MAX_NODES 100

typedef struct {
    int adjacencyMatrix[MAX_NODES][MAX_NODES];
    bool nodePresent[MAX_NODES];
    char data[MAX_NODES];
} Graph;
```

인접 행렬을 사용한 그래프 구현 예시

- 그래프 자료구조는 다음과 같은 정보를 가지는 데이터 타입
- create: 비어있는 그래프 데이터 (노드가 0개)를 생성 후 반환

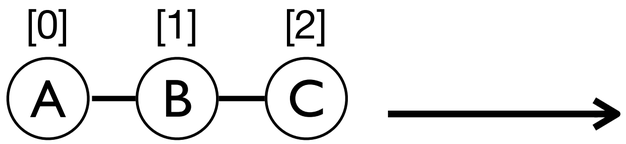
```c
#define MAX_NODES 100

typedef struct {
    int adjacencyMatrix[MAX_NODES][MAX_NODES];
    bool nodePresent[MAX_NODES];
    char data[MAX_NODES];
} Graph;
```

```
procedure create()
  graph ← allocateGraph()
  for i = 0 to MaxNodes()-1 do
    graph.nodePresent[i] = false
    for j = 0 to MaxNodes()-1 do
      graph.adjacencyMatrix[i][j] = 0;
   end for
  return graph
end procedure
```

insert node

- insert node: 그래프 데이터에 새로운 노드를 삽입함

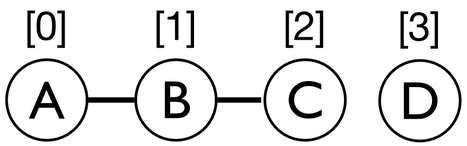


graph

D

data

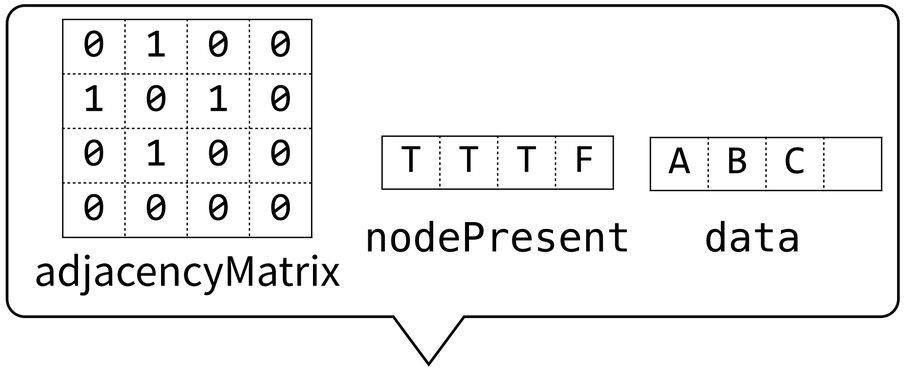
```
procedure insertNode(graph, data)
  idx ← -1
  for i = MaxNodes() to 0 do
    if (graph.nodePresent[i] = false) then
      idx ← i
    end if
  end for
  if idx = -1 then
    return
  end if
  graph.nodePresent[idx] ← true
  graph.data[idx] ← data
end procedure
```

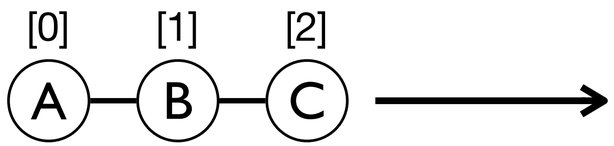


graph

insert node

- insert node: 그래프 데이터에 새로운 노드를 삽입함



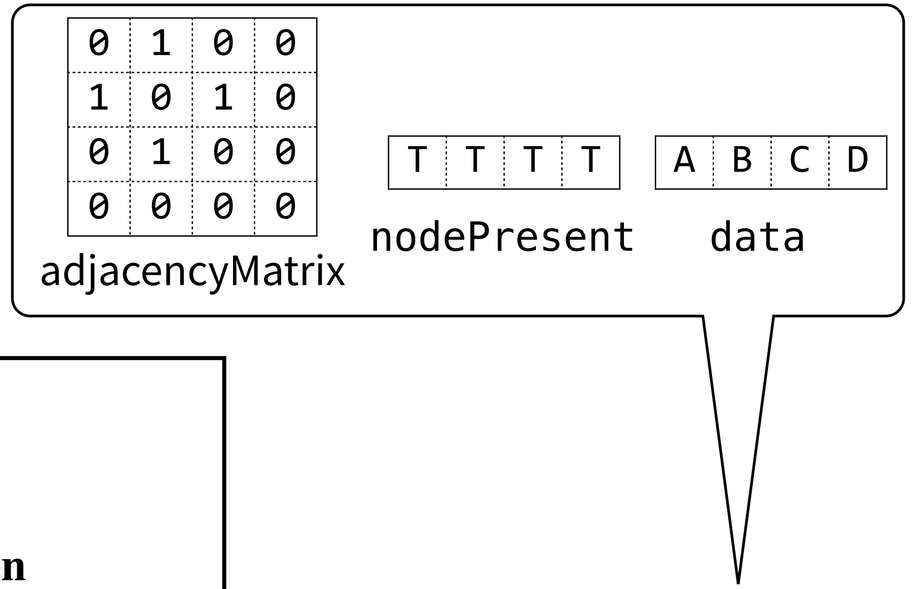


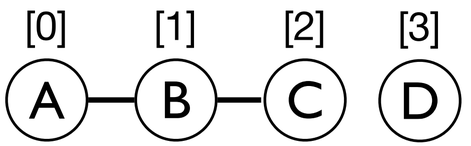
graph

D

data

```
procedure insertNode(graph, data)
  idx ← -1
  for i = MaxNodes() to 0 do
    if (graph.nodePresent[i] = false) then
      idx ← i
    end if
  end for
  if idx = -1 then
    return
  end if
  graph.nodePresent[idx] ← true
  graph.data[idx] ← data
end procedure
```

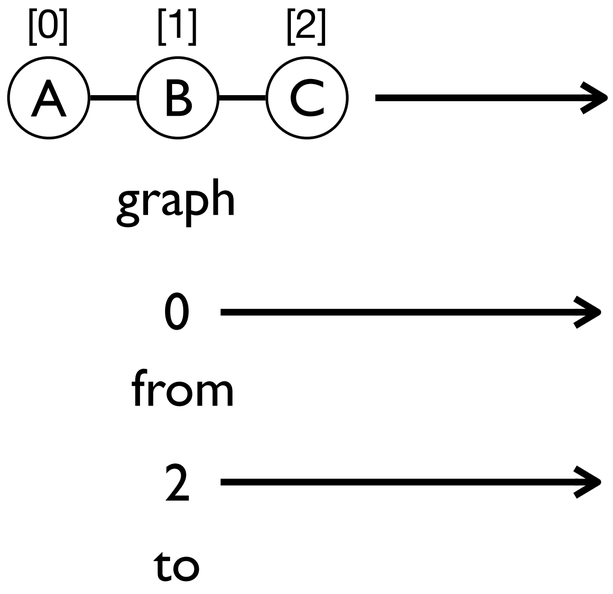




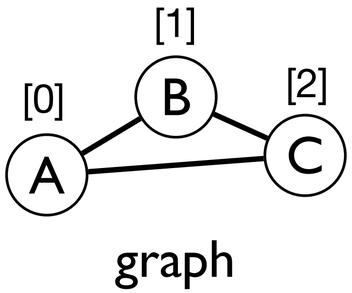
graph

insert edge

- insert edge: 그래프 데이터에 새로운 간선을 추가함




```
procedure insertEdge(graph, from, to)
  if invalidIndices(graph, from, to) then
    return
  end if
  graph.adjacencyMatrix[from][to] ← 1
  if isUndirected() then
    graph.adjacencyMatrix[to][from] ← 1
  end if
end procedure
```



insert edge

insert edge: 그래프 데이터에 새로운 간선을 추가함

0	1	0	0
1	0	1	0
0	1	0	0
0	0	0	0

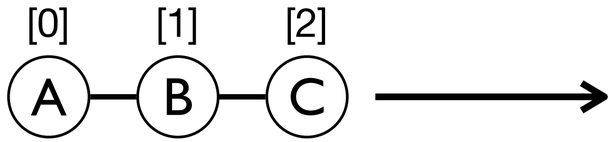
adjacencyMatrix

T	T	T	F

nodePresent

A	B	C	

data



graph

0

from

2

to

```
procedure insertEdge(graph, from, to)
  if invalidIndices(graph, from, to) then
    return
  end if
  graph.adjacencyMatrix[from][to] ← 1
  if isUndirected() then
    graph.adjacencyMatrix[to][from] ← 1
  end if
end procedure
```

0	1	1	0
1	0	1	0
1	1	0	0
0	0	0	0

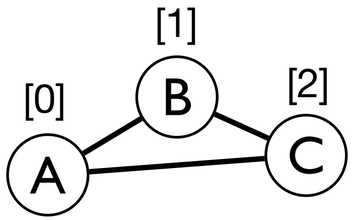
adjacencyMatrix

T	T	T	F

nodePresent

A	B	C	

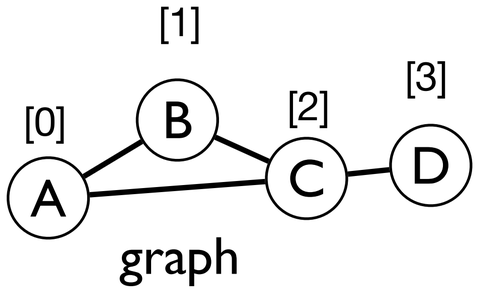
data



graph

delete node

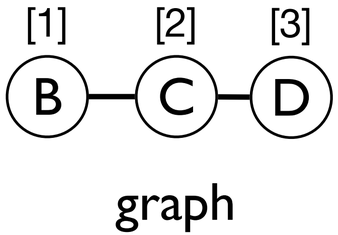
- delete node: 그래프 데이터에 있는 노드를 삭제함



0

nodeIdx

```
procedure deleteNode(graph, nodeIdx)
  if invalidIndex(graph, nodeIdx) then
    return
  end if
  for i = 0 to MaxNodes() do
    graph.adjacencyMatrix[nodeIdx][i] ← 0
    graph.adjacencyMatrix[i][nodeIdx] ← 0
  end for
  graph.nodePresent[nodeIdx] ← false
end procedure
```



delete node

- 있는 노드를 삭제함

0	1	1	0
1	0	1	0
1	1	0	1
0	0	1	0

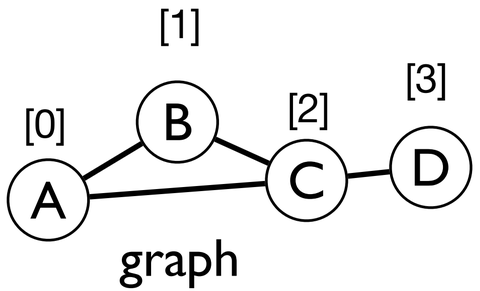
adjacencyMatrix

T	T	T	T

nodePresent

A	B	C	D

data



graph

0

nodeIdx

```
procedure deleteNode(graph, nodeIdx)
  if invalidIndex(graph, nodeIdx) then
    return
  end if
  for i = 0 to MaxNodes() do
    graph.adjacencyMatrix[nodeIdx][i] ← 0
    graph.adjacencyMatrix[i][nodeIdx] ← 0
  end for
  graph.nodePresent[nodeIdx] ← false
end procedure
```

0	0	0	0
0	0	1	0
0	1	0	1
0	0	1	0

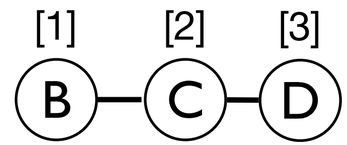
adjacencyMatrix

F	T	T	T

nodePresent

A	B	C	D

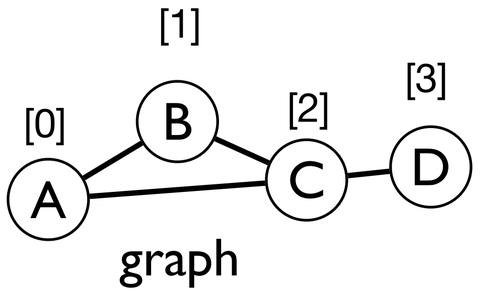
data



graph

delete edge

- delete edge: 그래프 데이터에서 간선을 삭제함



graph

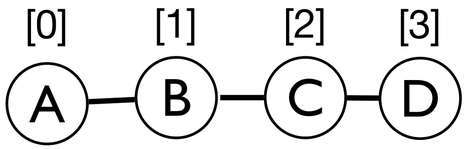
0

from

2

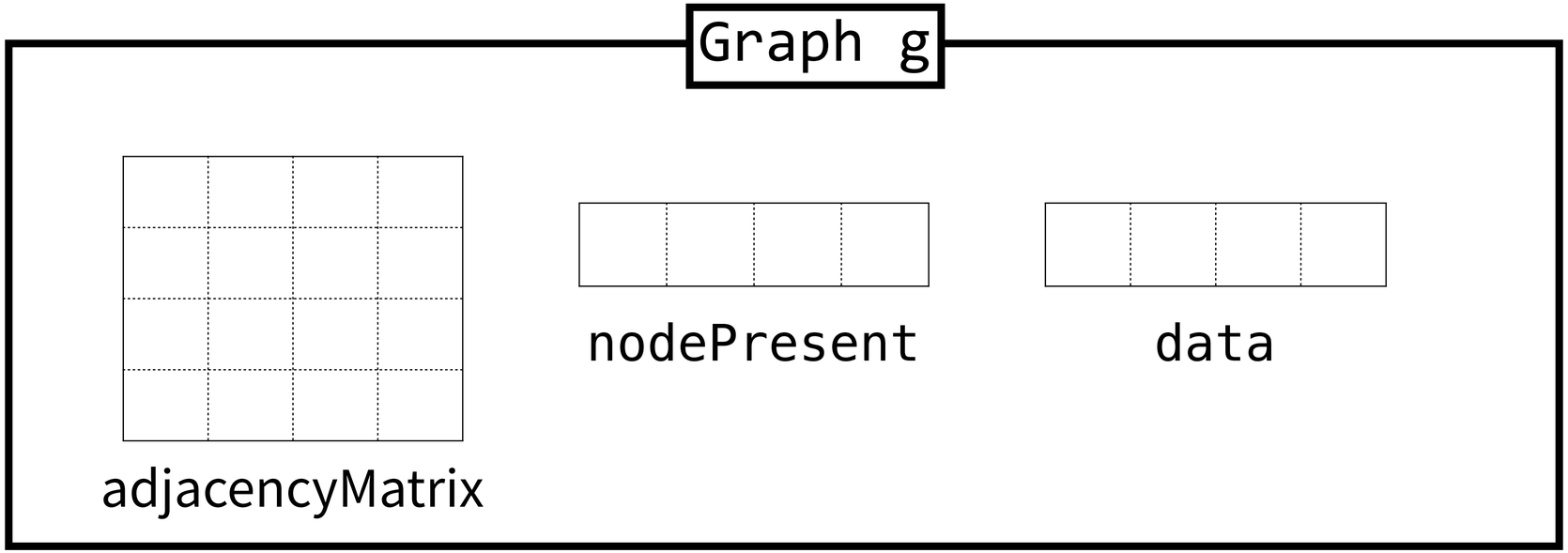
to

```
procedure deleteEdge(graph, from, to)
  if invalidIndices(graph, from, to) then
    return
  end if
  graph.adjacencyMatrix[from][to] ← 0
  if isUndirected() then
    graph.adjacencyMatrix[to][from] ← 0
  end if
end procedure
```



graph

Example




```
create() → insertNode(g,A) → insertNode(g,B)
                             insertEdge(g,0,1) → insertNode(g,C)
                                                 insertEdge(g,1,2)
                                                          ↓
deleteNode(g,2) ← deleteEdge(g,1,2) ← deleteNode(g,0)
```

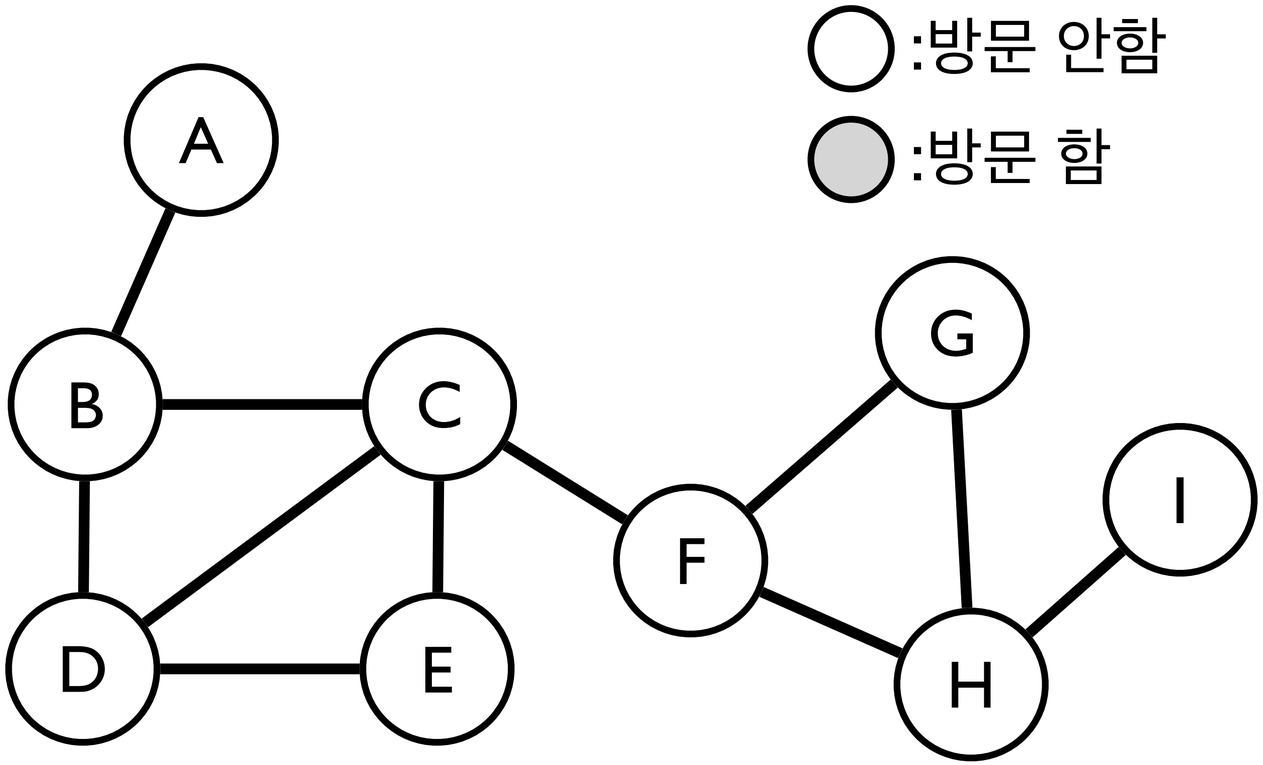
search

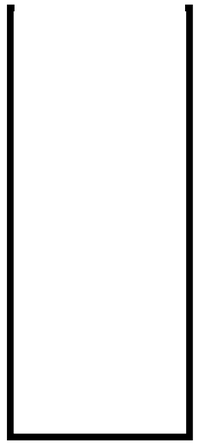
- search: 그래프 데이터에서 주어진 키를 가진 노드를 반환함
- 두가지 대표적인 방법이 있음
- 깊이 우선 탐색 (depth first search)
    - 스택 자료구조를 사용하여 그래프를 순회함
- 넓이 우선 탐색 (breadth first search)
    - 스택 자료구조를 사용하여 그래프를 순회함

Depth First Search (깊이 우선 탐색)

- 깊이우선 탐색(DFS)는 특정 경로를 끝까지 탐색 후 경로가 끝나면 돌아와 다른 경로를 탐색함

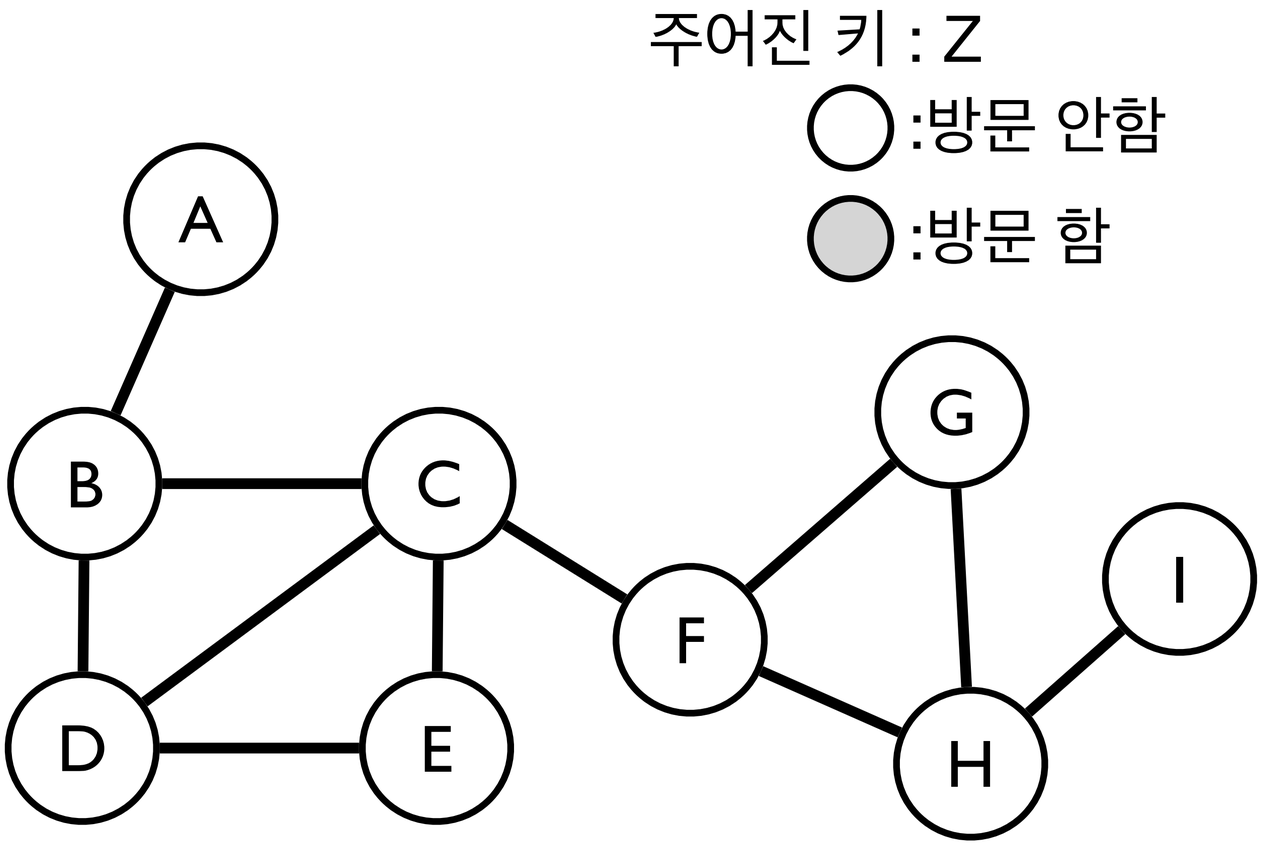
주어진 키 : Z

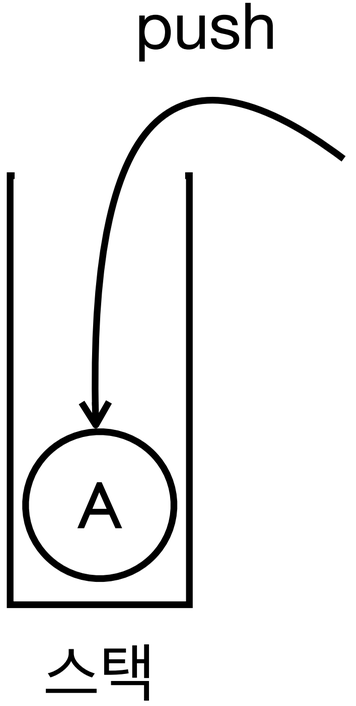




스택

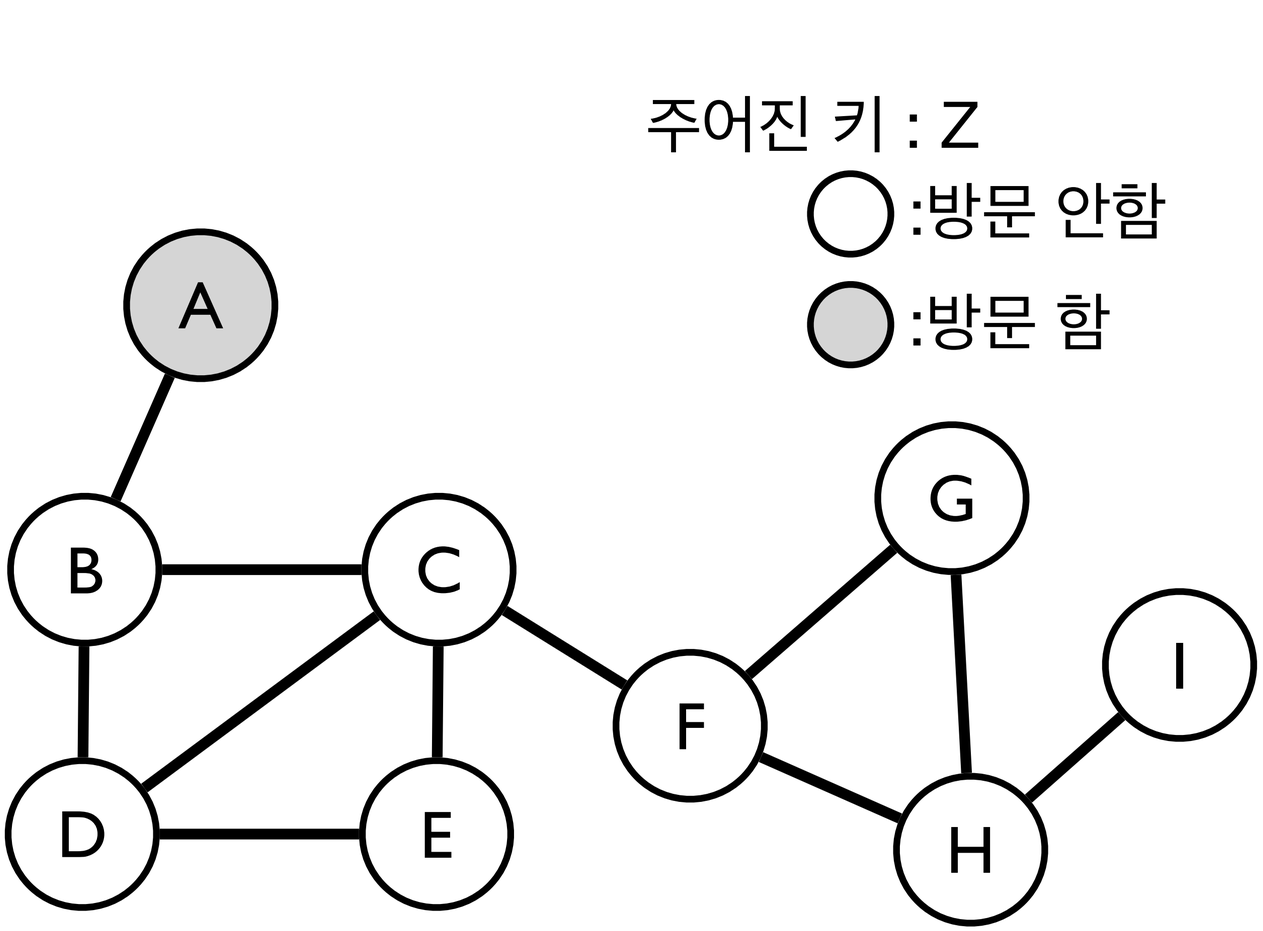
(1) 시작 노드를 스택에 push 함

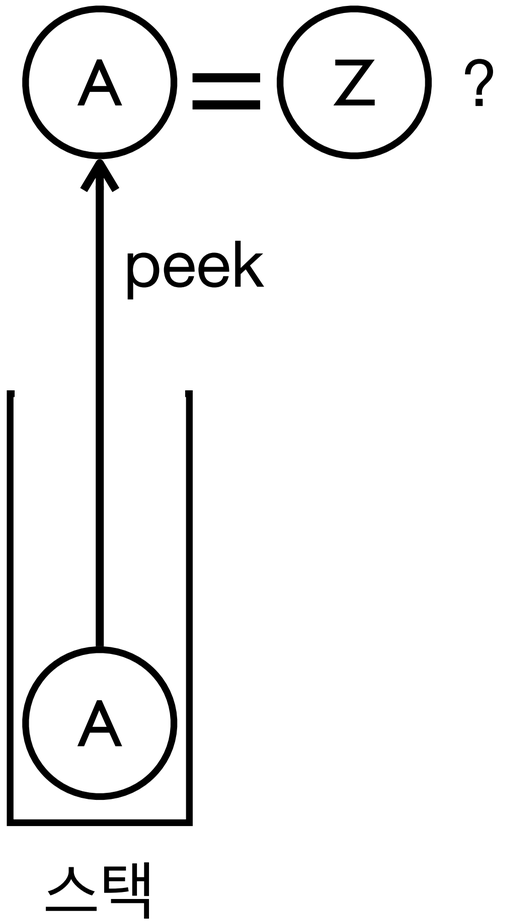




(1) 시작 노드를 스택에 push 함

(2) 스택의 top노드를 방문해 찾고자 하는 노드인지 확인함 (찾았을 경우 해당 노드를 반환 후 종료).

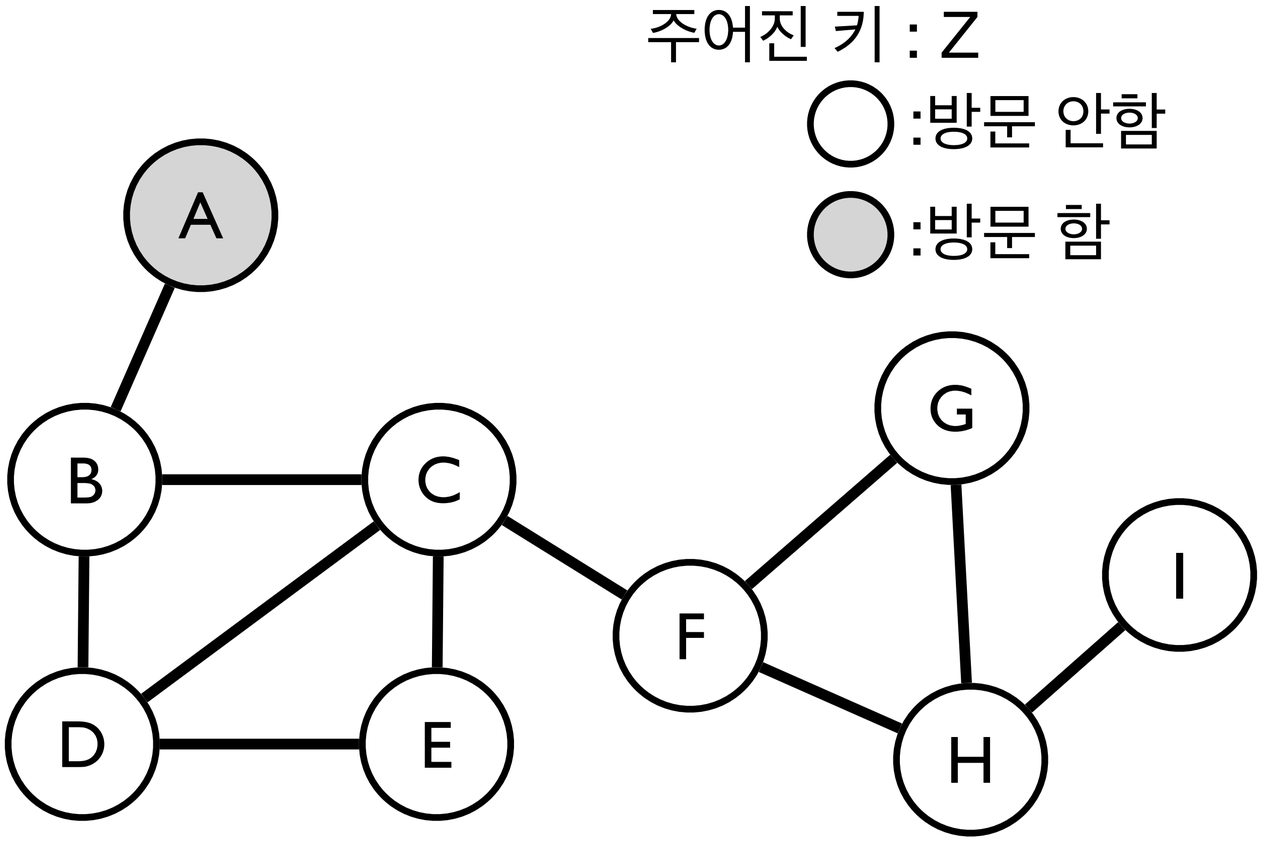


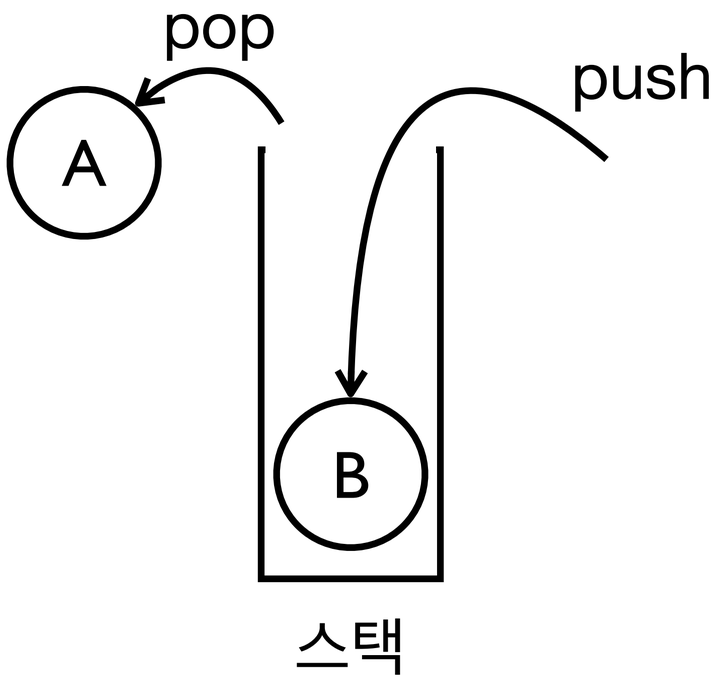


(1) 시작 노드를 스택에 push 함

(2) 스택의 top노드를 방문해 찾고자 하는 노드인지 확인함 (찾았을 경우 해당 노드를 반환 후 종료).

(3) 스택을 pop하고 pop된 노드의 인접 노드 중 방문하지 않았으면서 스택에 없는 노드들을 스택에 push함



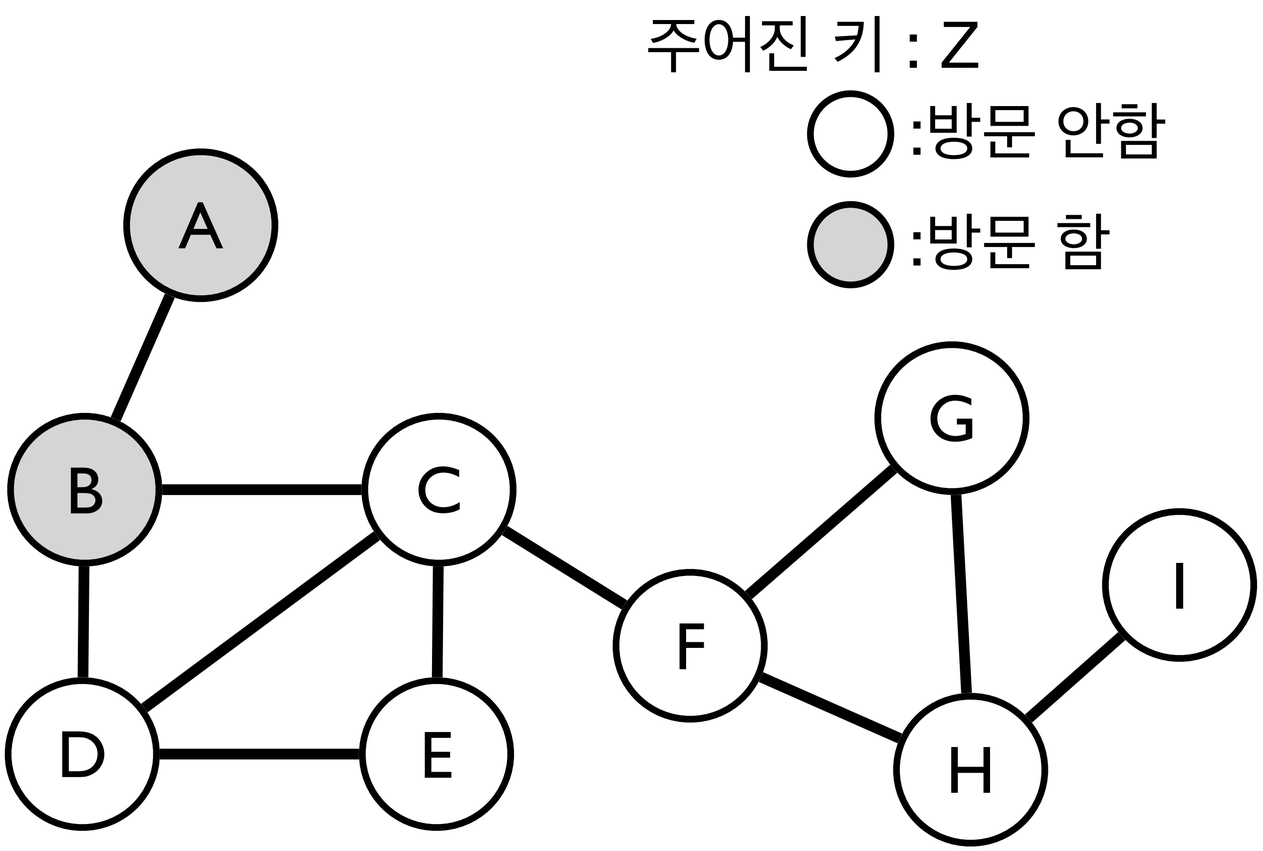


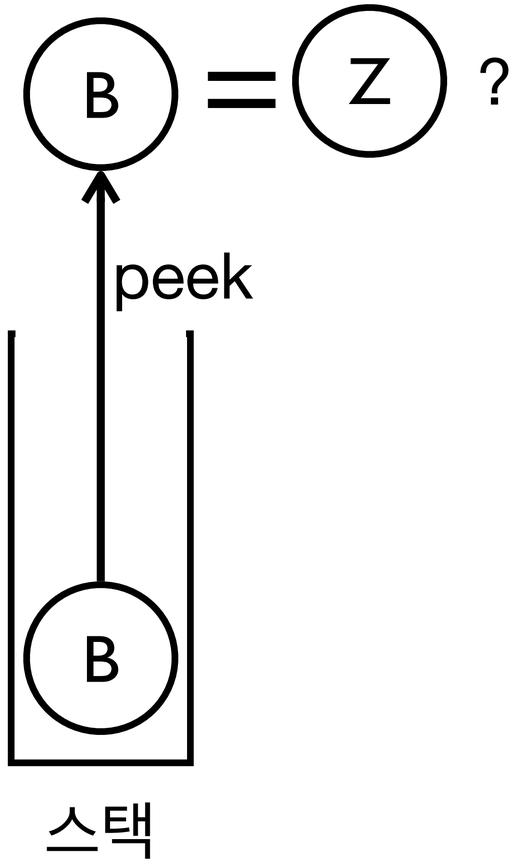
(1) 시작 노드를 스택에 push 함

(2) 스택의 top노드를 방문해 찾고자 하는 노드인지 확인함 (찾았을 경우 해당 노드를 반환 후 종료).

(3) 스택을 pop하고 pop된 노드의 인접 노드 중 방문하지 않았으면서 스택에 없는 노드들을 스택에 push함

(4) 스택이 빌 때까지 (2), (3)을 반복함



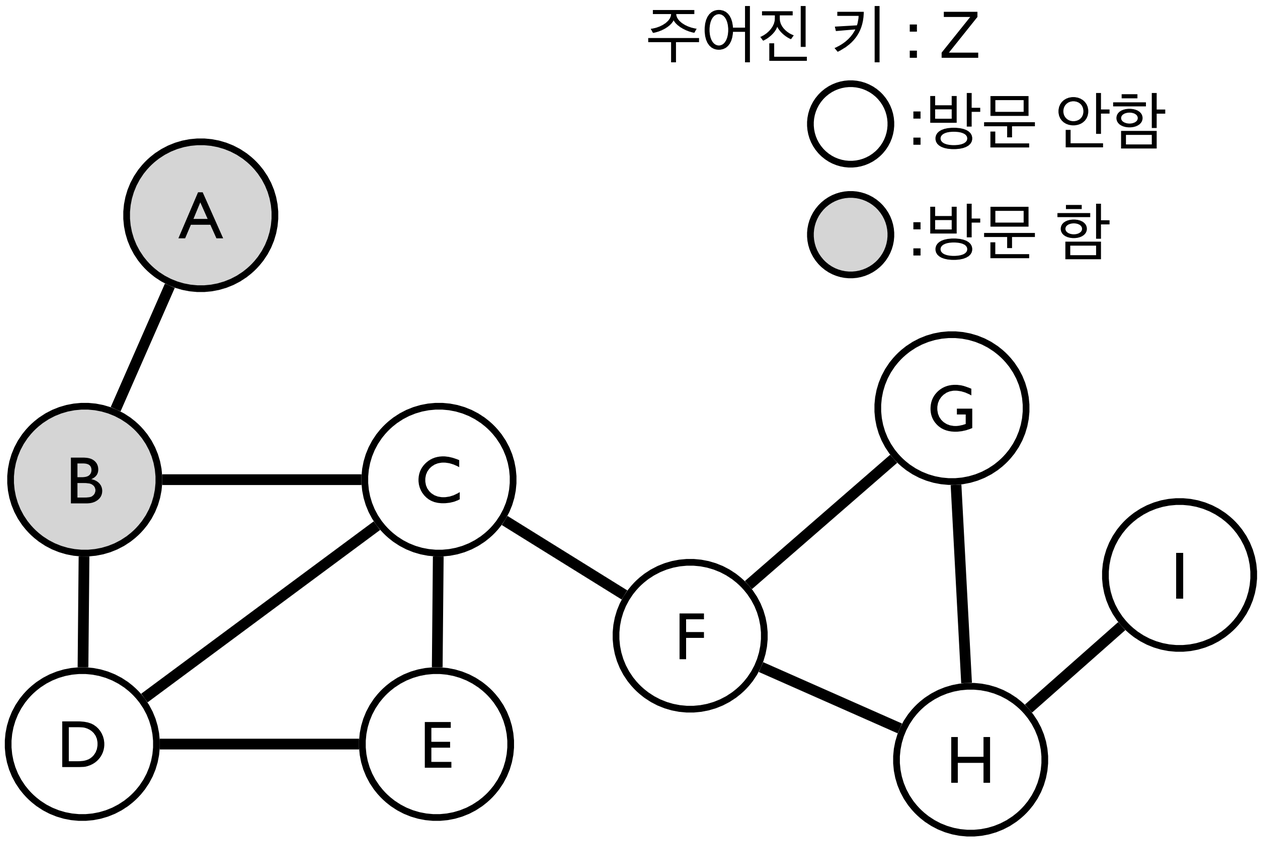


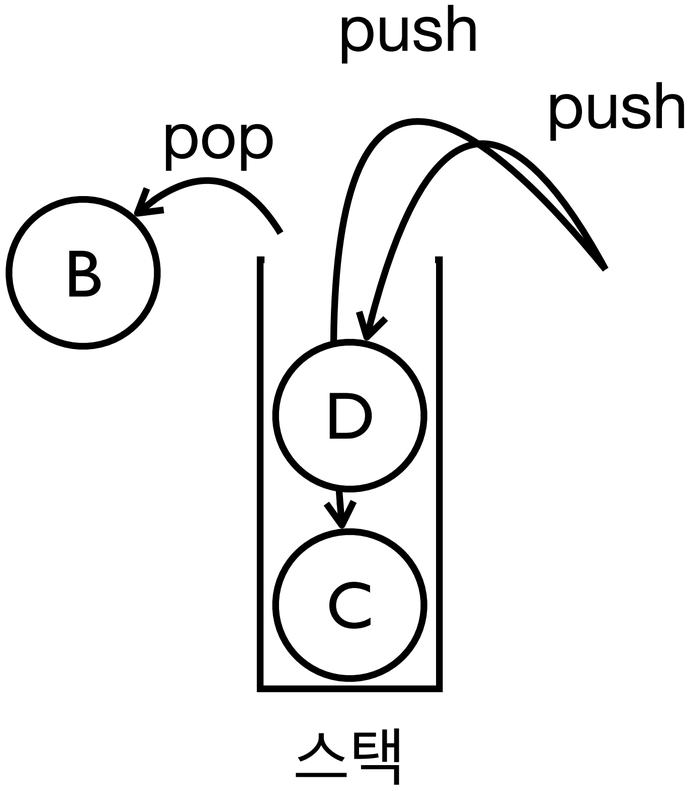
(1) 시작 노드를 스택에 push 함

(2) 스택의 top노드를 방문해 찾고자 하는 노드인지 확인함 (찾았을 경우 해당 노드를 반환 후 종료).

(3) 스택을 pop하고 pop된 노드의 인접 노드 중 방문하지 않았으면서 스택에 없는 노드들을 스택에 push함

(4) 스택이 빌 때까지 (2), (3)을 반복함



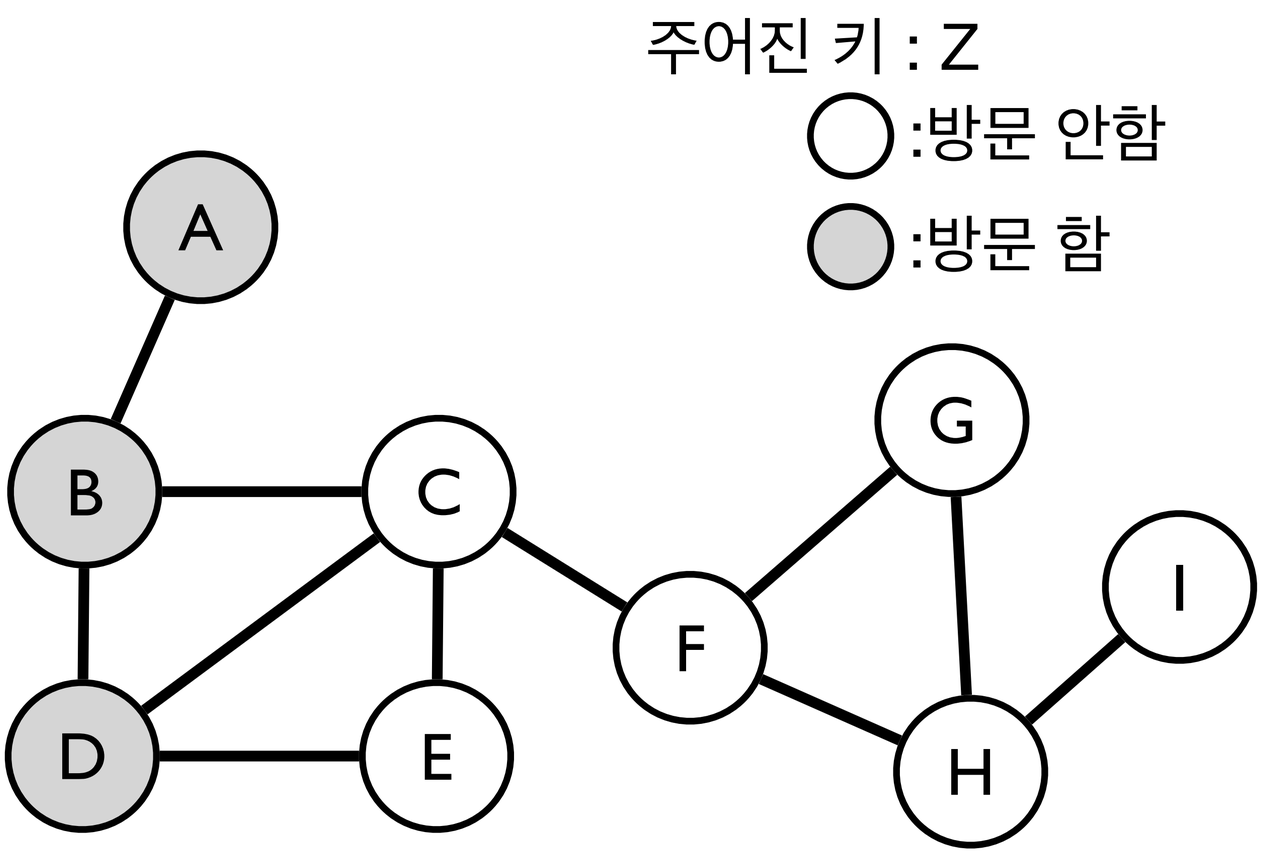


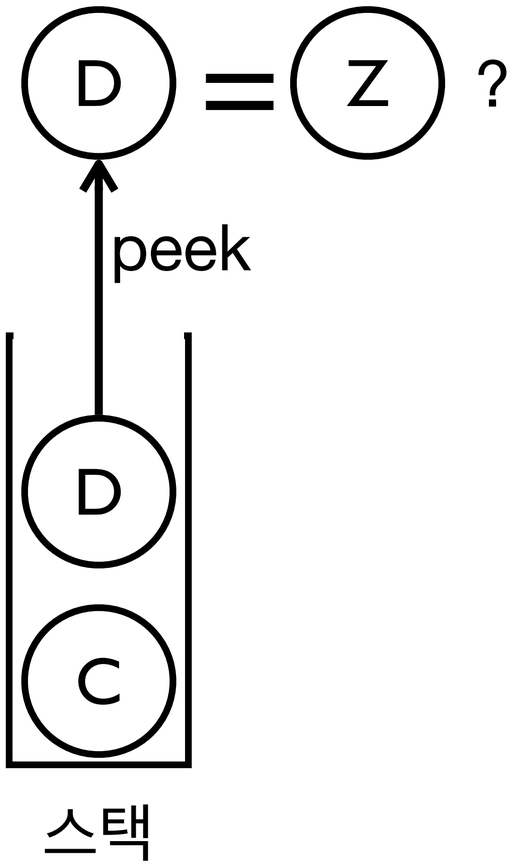
(1) 시작 노드를 스택에 push 함

(2) 스택의 top노드를 방문해 찾고자 하는 노드인지 확인함 (찾았을 경우 해당 노드를 반환 후 종료).

(3) 스택을 pop하고 pop된 노드의 인접 노드 중 방문하지 않았으면서 스택에 없는 노드들을 스택에 push함

(4) 스택이 빌 때까지 (2), (3)을 반복함



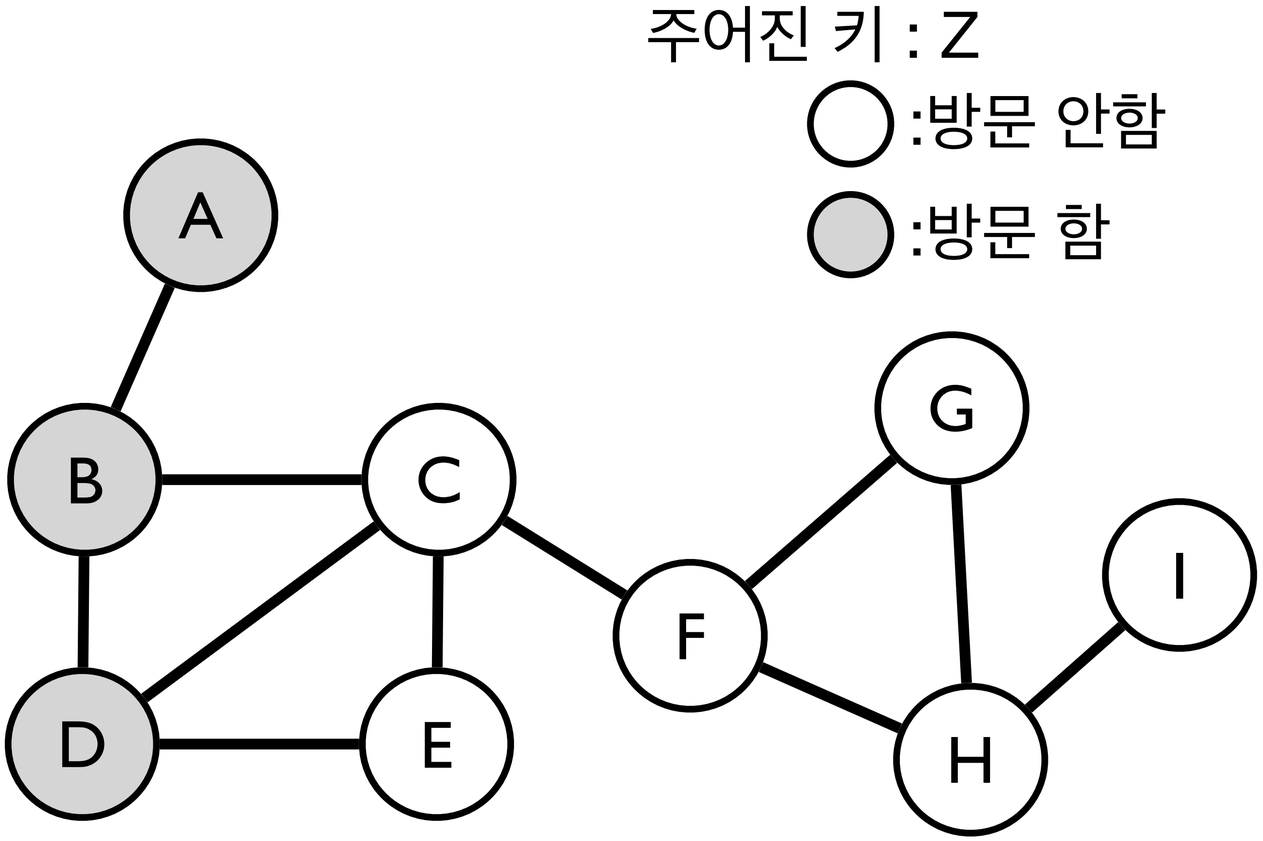


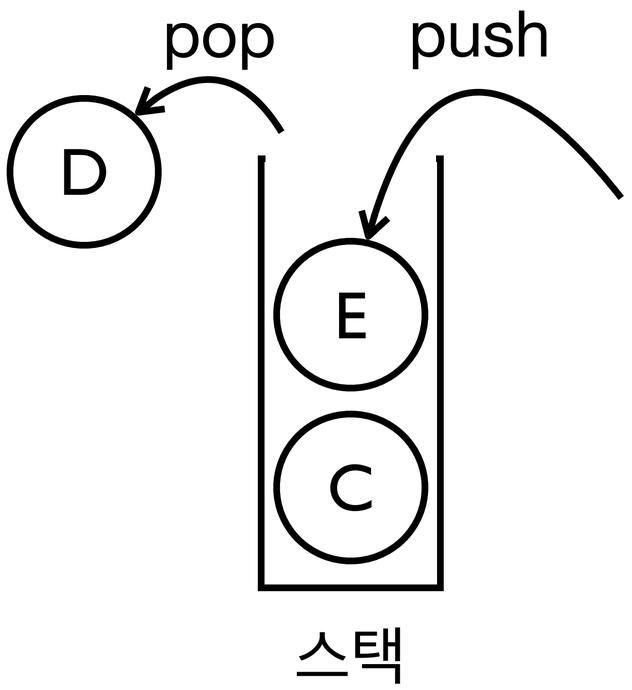
(1) 시작 노드를 스택에 push 함

(2) 스택의 top노드를 방문해 찾고자 하는 노드인지 확인함 (찾았을 경우 해당 노드를 반환 후 종료).

(3) 스택을 pop하고 pop된 노드의 인접 노드 중 방문하지 않았으면서 스택에 없는 노드들을 스택에 push함

(4) 스택이 빌 때까지 (2), (3)을 반복함



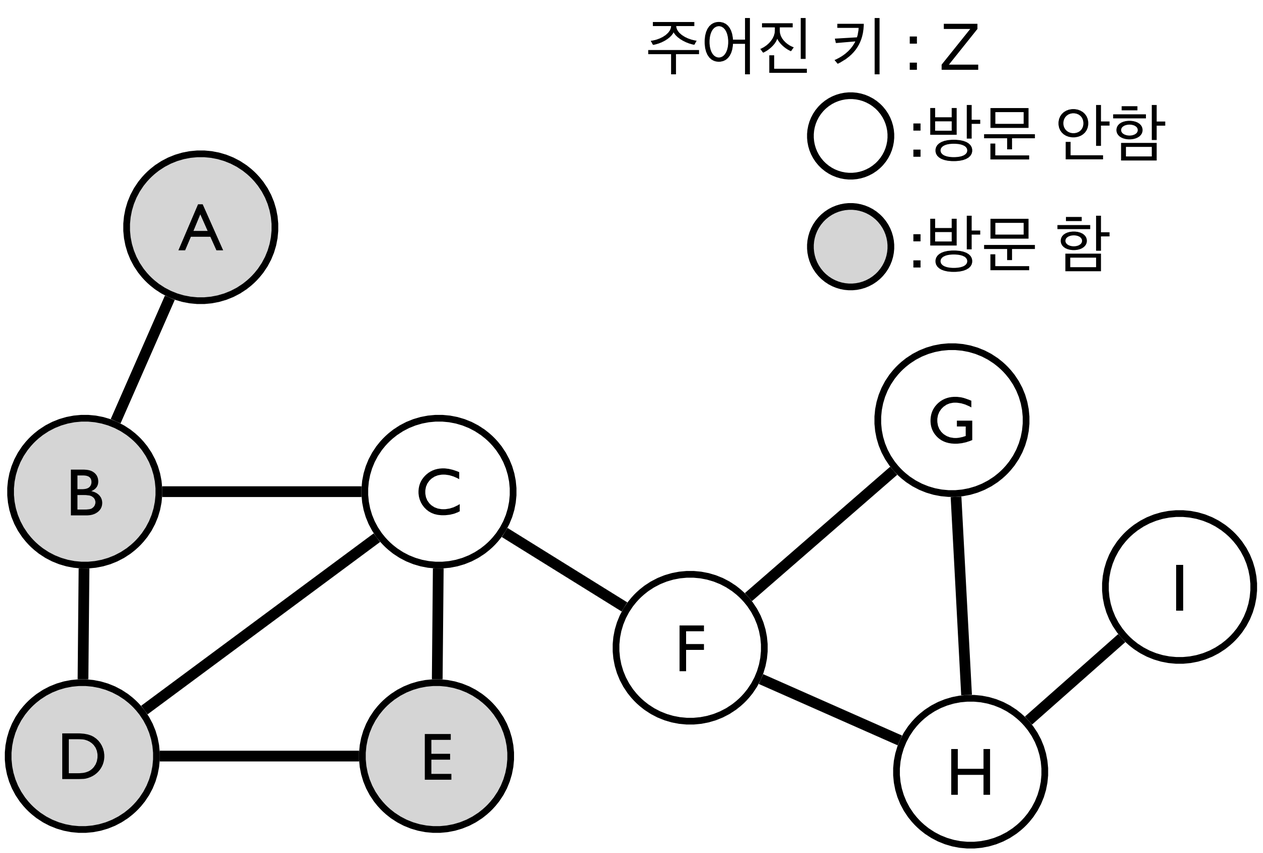


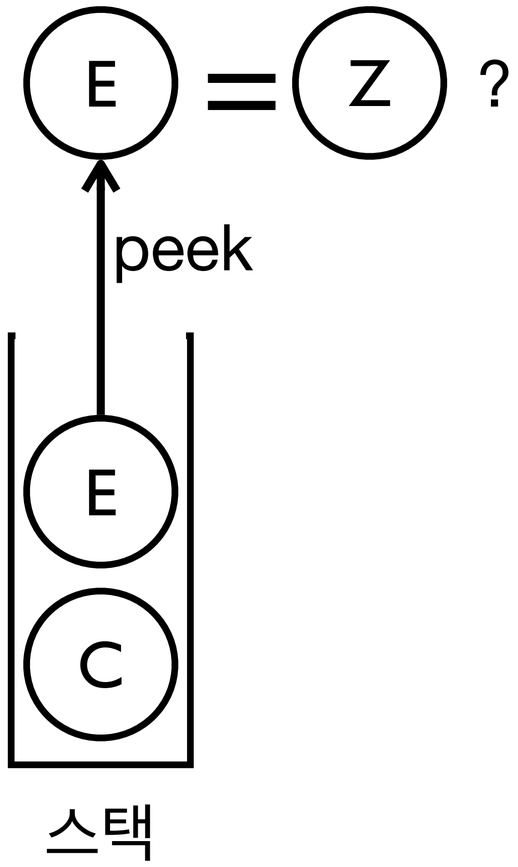
(1) 시작 노드를 스택에 push 함

(2) 스택의 top노드를 방문해 찾고자 하는 노드인지 확인함 (찾았을 경우 해당 노드를 반환 후 종료).

(3) 스택을 pop하고 pop된 노드의 인접 노드 중 방문하지 않았으면서 스택에 없는 노드들을 스택에 push함

(4) 스택이 빌 때까지 (2), (3)을 반복함



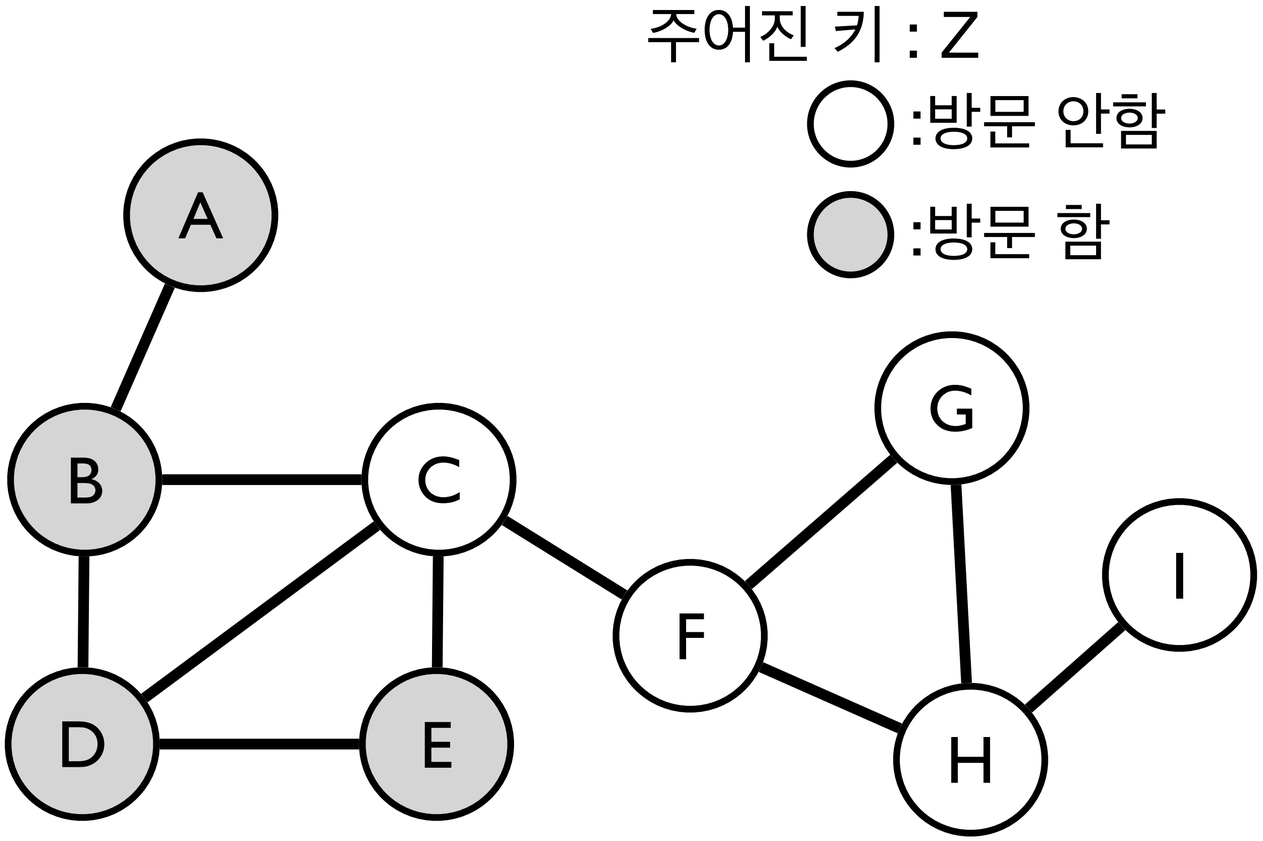


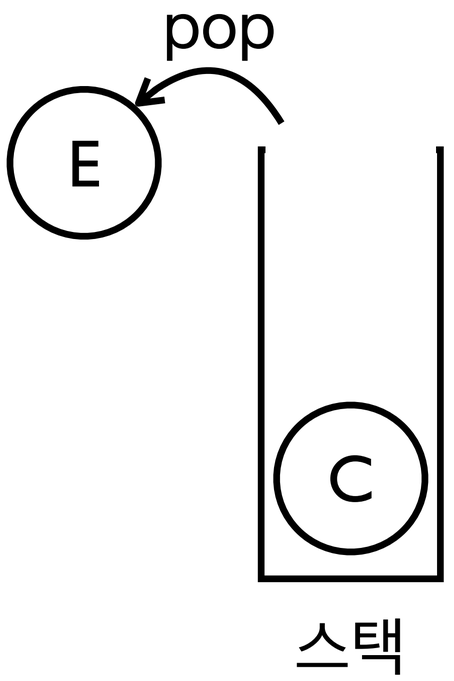
(1) 시작 노드를 스택에 push 함

(2) 스택의 top노드를 방문해 찾고자 하는 노드인지 확인함 (찾았을 경우 해당 노드를 반환 후 종료).

(3) 스택을 pop하고 pop된 노드의 인접 노드 중 방문하지 않았으면서 스택에 없는 노드들을 스택에 push함

(4) 스택이 빌 때까지 (2), (3)을 반복함



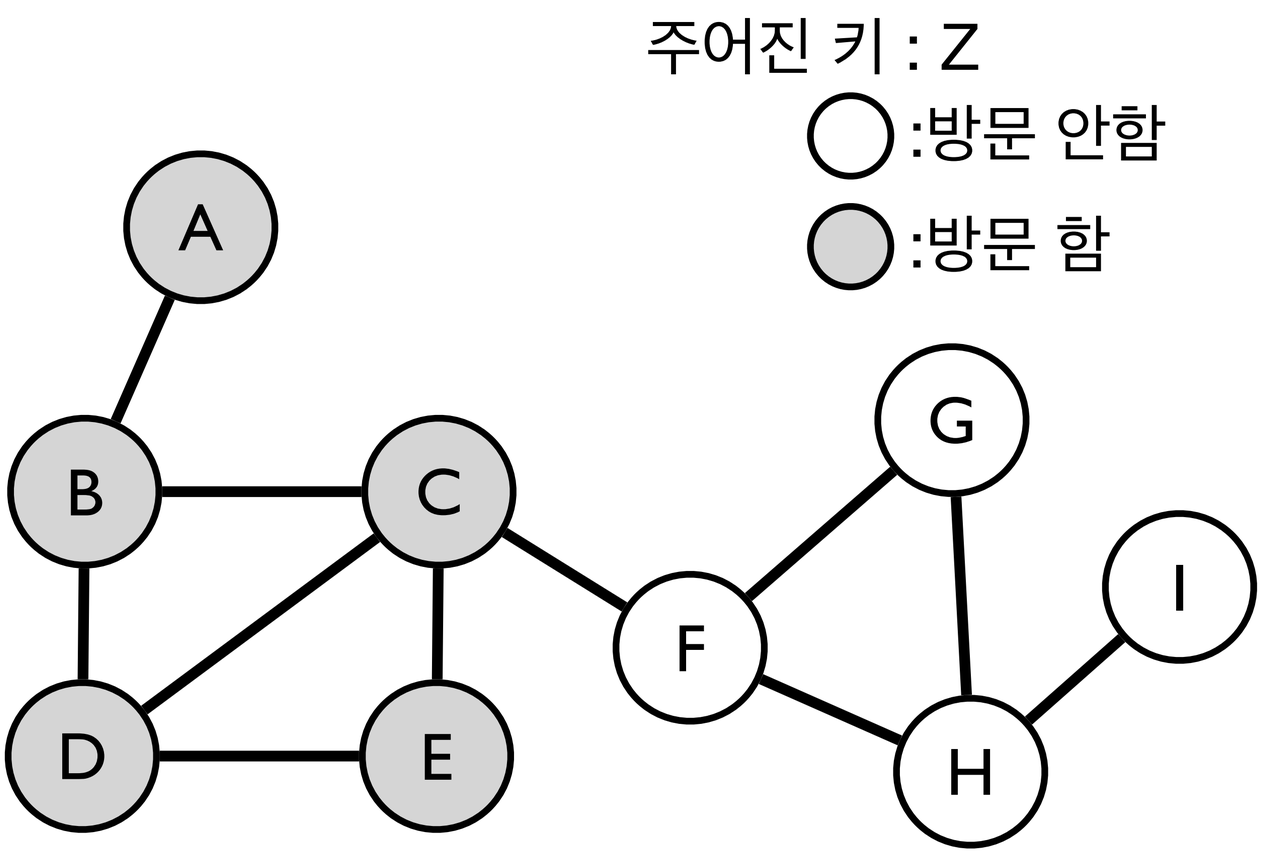


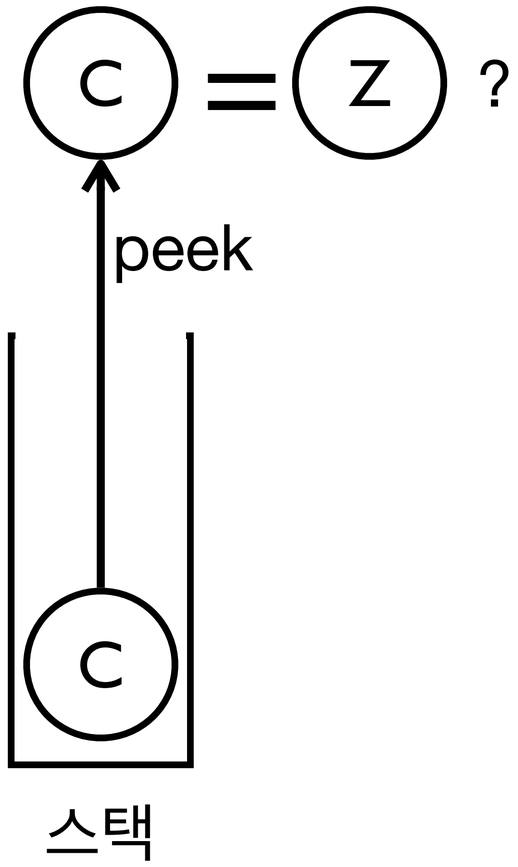
(1) 시작 노드를 스택에 push 함

(2) 스택의 top노드를 방문해 찾고자 하는 노드인지 확인함 (찾았을 경우 해당 노드를 반환 후 종료).

(3) 스택을 pop하고 pop된 노드의 인접 노드 중 방문하지 않았으면서 스택에 없는 노드들을 스택에 push함

(4) 스택이 빌 때까지 (2), (3)을 반복함



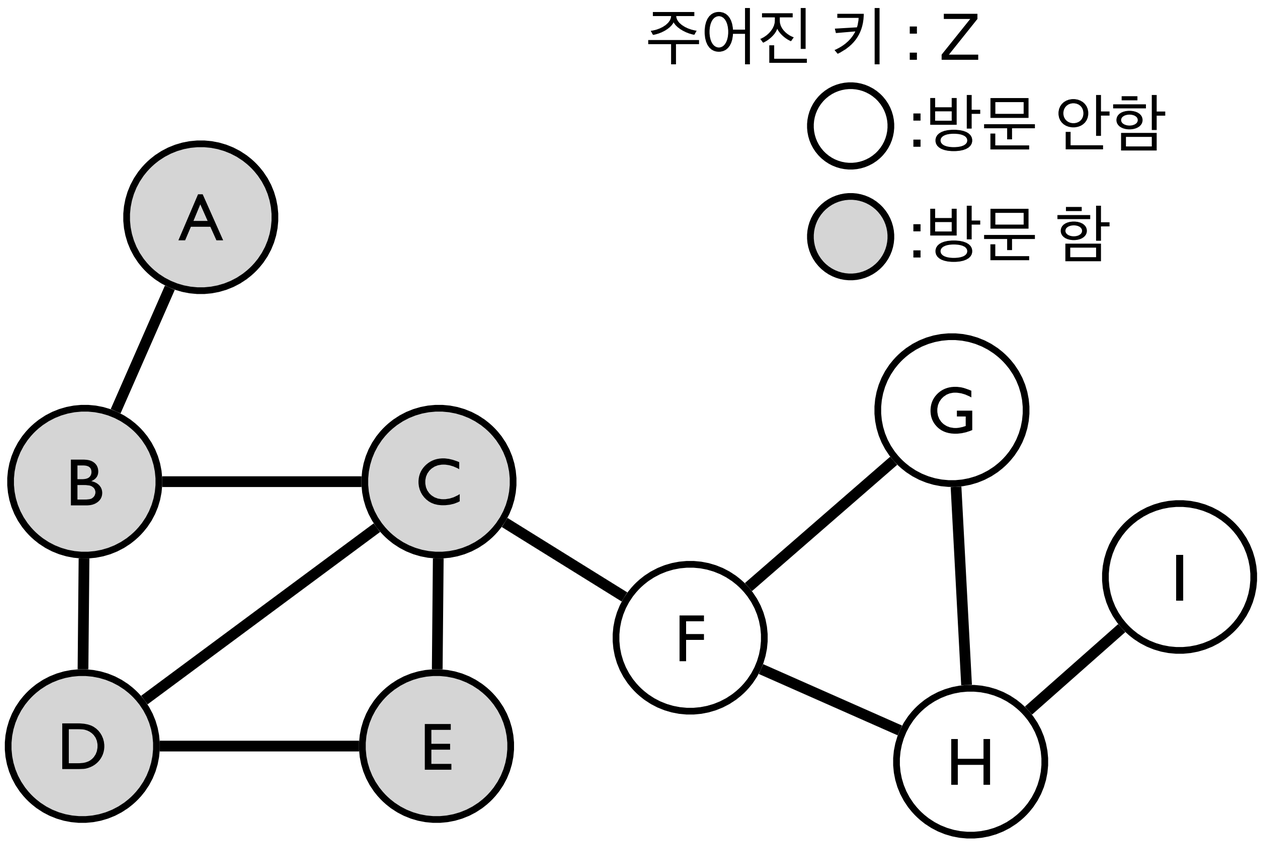


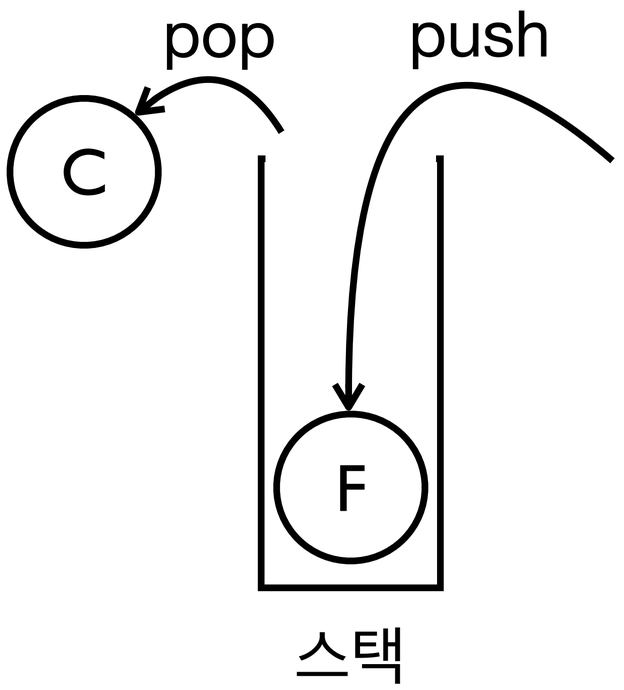
(1) 시작 노드를 스택에 push 함

(2) 스택의 top노드를 방문해 찾고자 하는 노드인지 확인함 (찾았을 경우 해당 노드를 반환 후 종료).

(3) 스택을 pop하고 pop된 노드의 인접 노드 중 방문하지 않았으면서 스택에 없는 노드들을 스택에 push함

(4) 스택이 빌 때까지 (2), (3)을 반복함



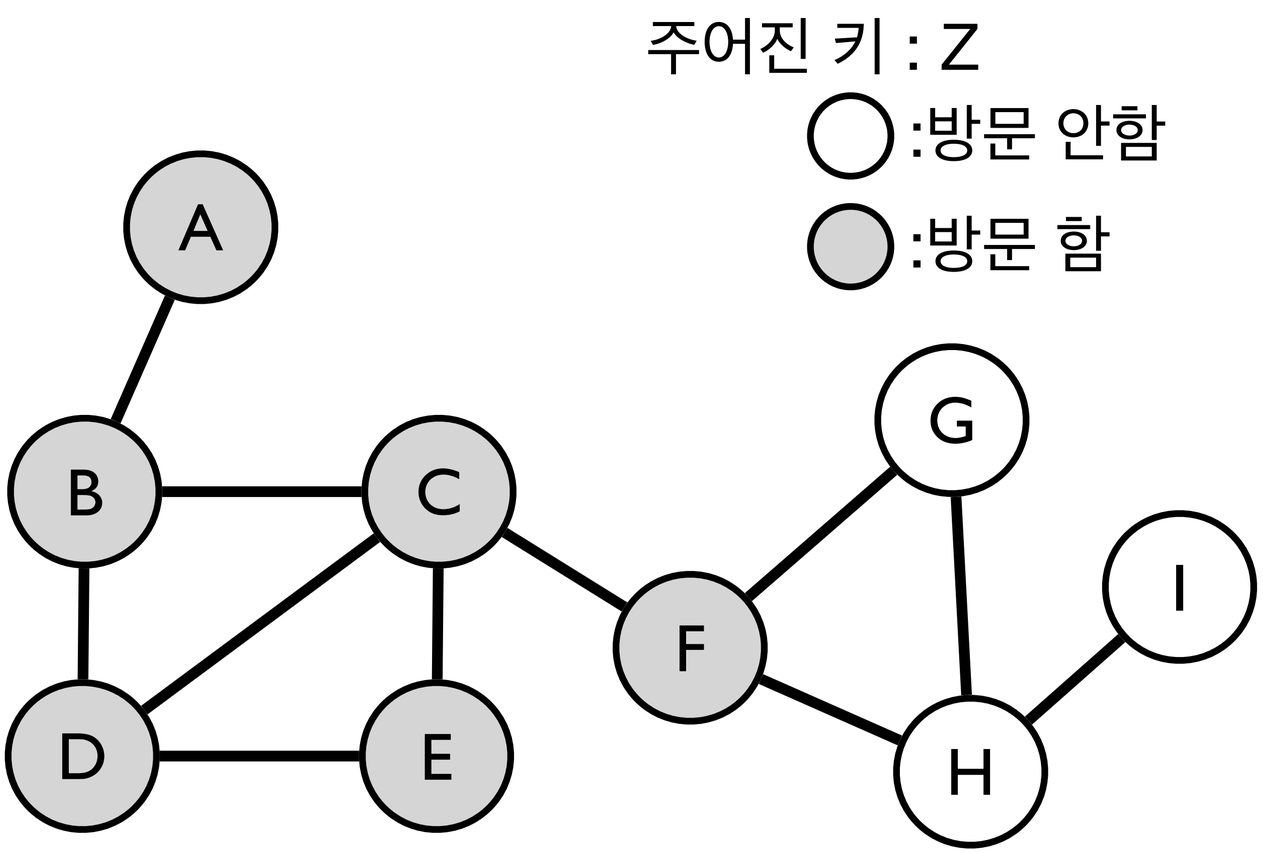


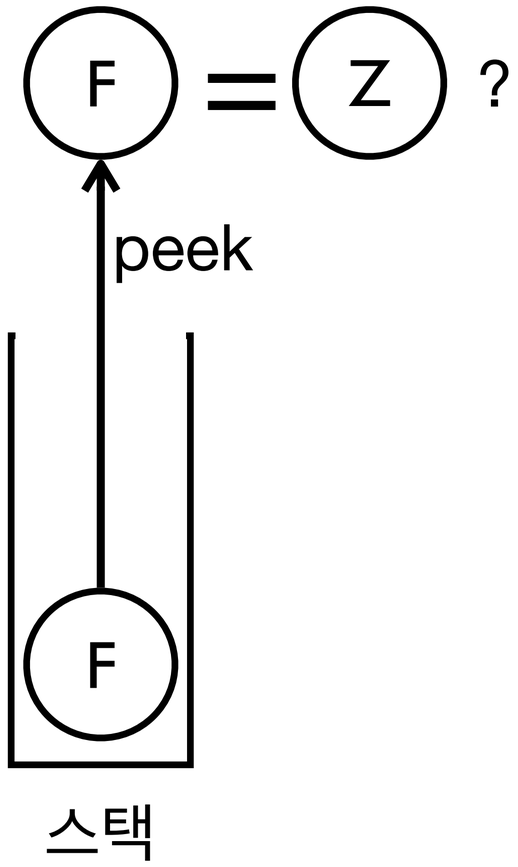
(1) 시작 노드를 스택에 push 함

(2) 스택의 top노드를 방문해 찾고자 하는 노드인지 확인함 (찾았을 경우 해당 노드를 반환 후 종료).

(3) 스택을 pop하고 pop된 노드의 인접 노드 중 방문하지 않았으면서 스택에 없는 노드들을 스택에 push함

(4) 스택이 빌 때까지 (2), (3)을 반복함



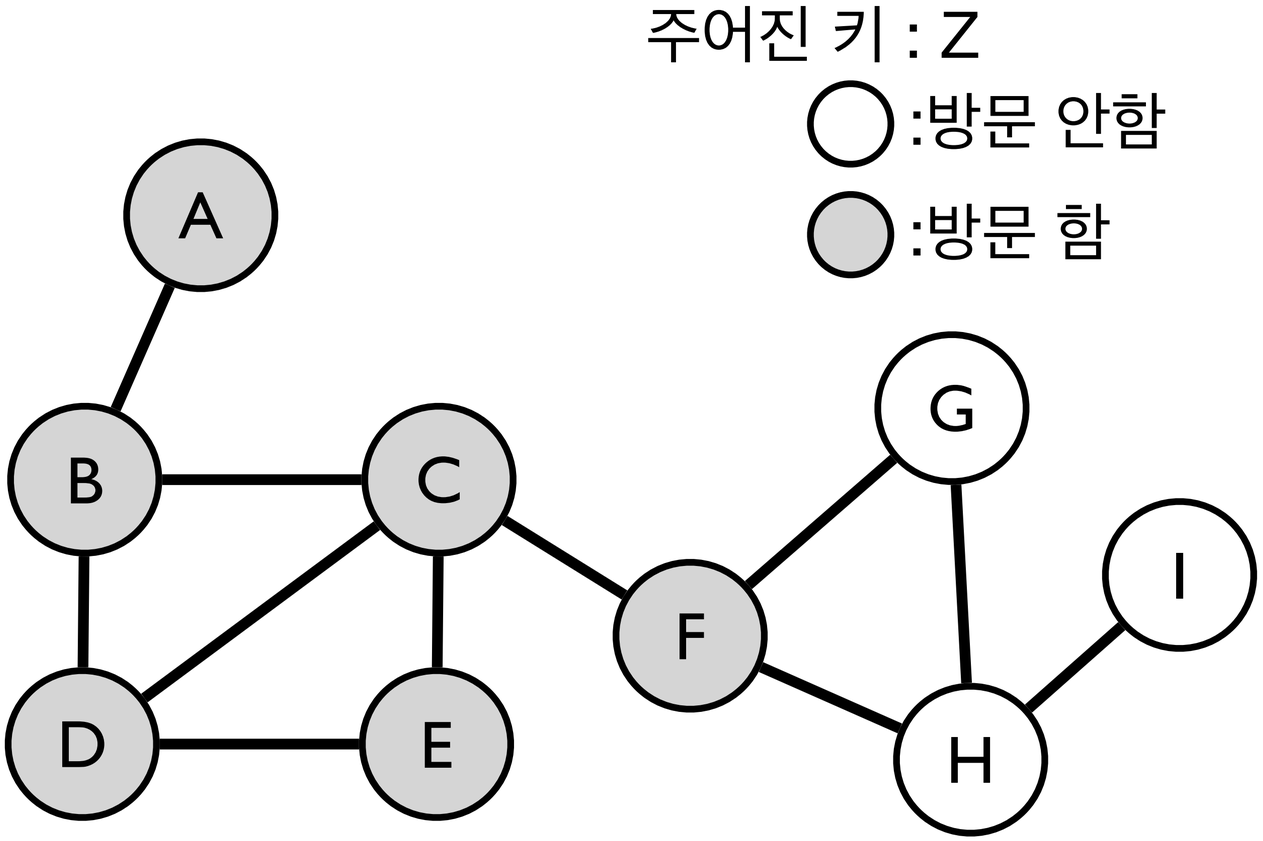


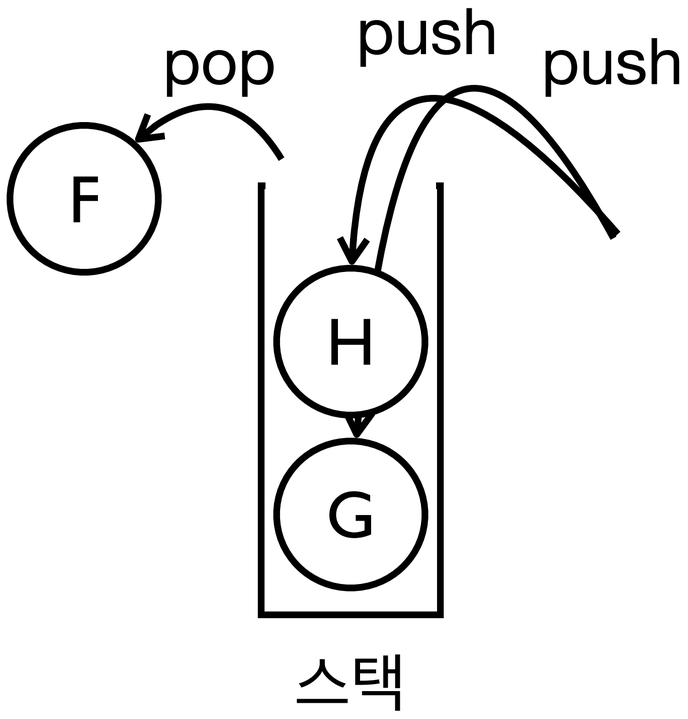
(1) 시작 노드를 스택에 push 함

(2) 스택의 top노드를 방문해 찾고자 하는 노드인지 확인함 (찾았을 경우 해당 노드를 반환 후 종료).

(3) 스택을 pop하고 pop된 노드의 인접 노드 중 방문하지 않았으면서 스택에 없는 노드들을 스택에 push함

(4) 스택이 빌 때까지 (2), (3)을 반복함



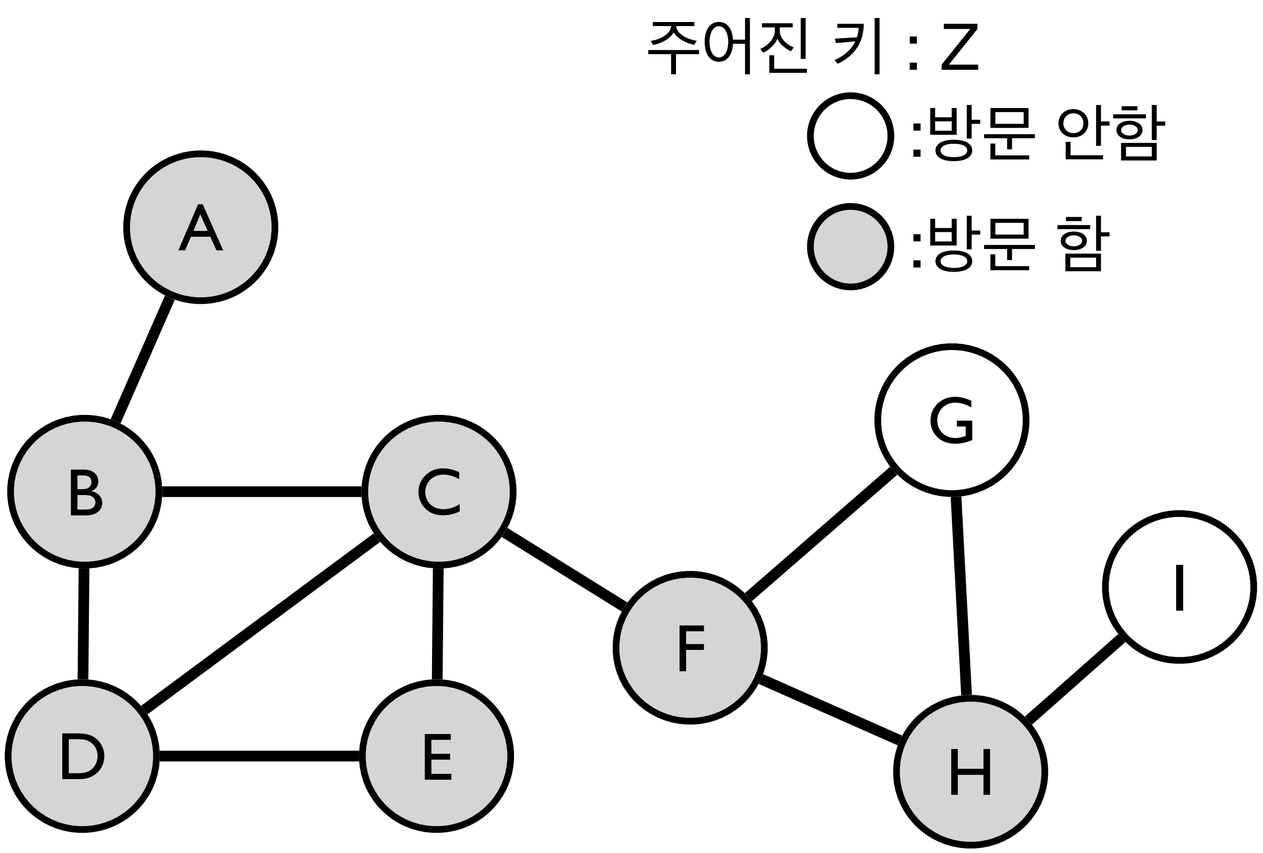


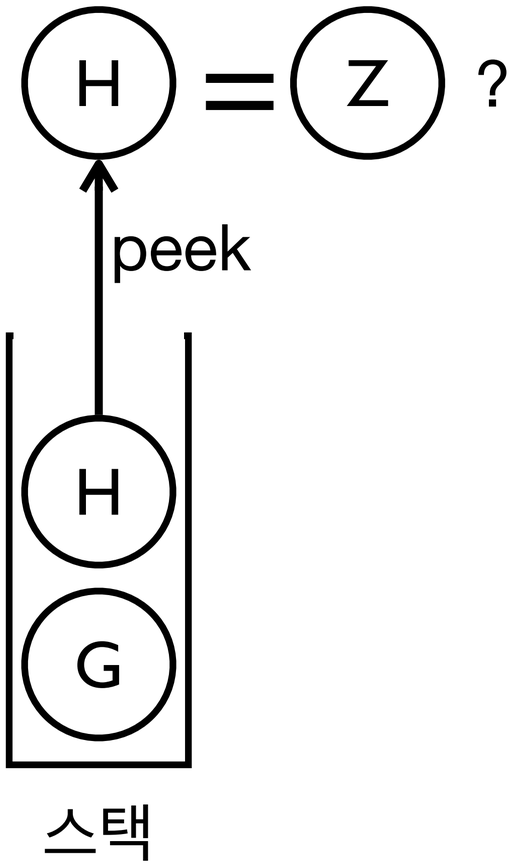
(1) 시작 노드를 스택에 push 함

(2) 스택의 top노드를 방문해 찾고자 하는 노드인지 확인함 (찾았을 경우 해당 노드를 반환 후 종료).

(3) 스택을 pop하고 pop된 노드의 인접 노드 중 방문하지 않았으면서 스택에 없는 노드들을 스택에 push함

(4) 스택이 빌 때까지 (2), (3)을 반복함



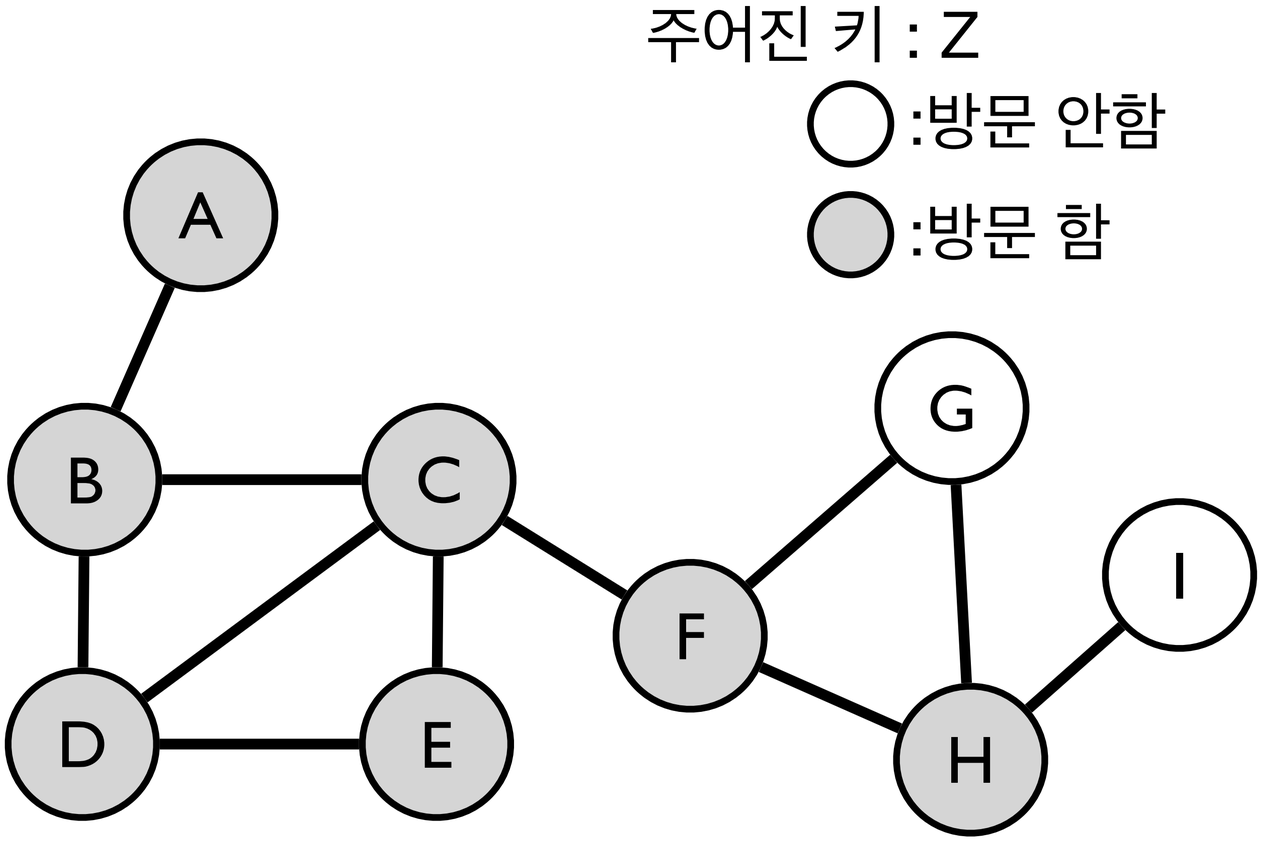


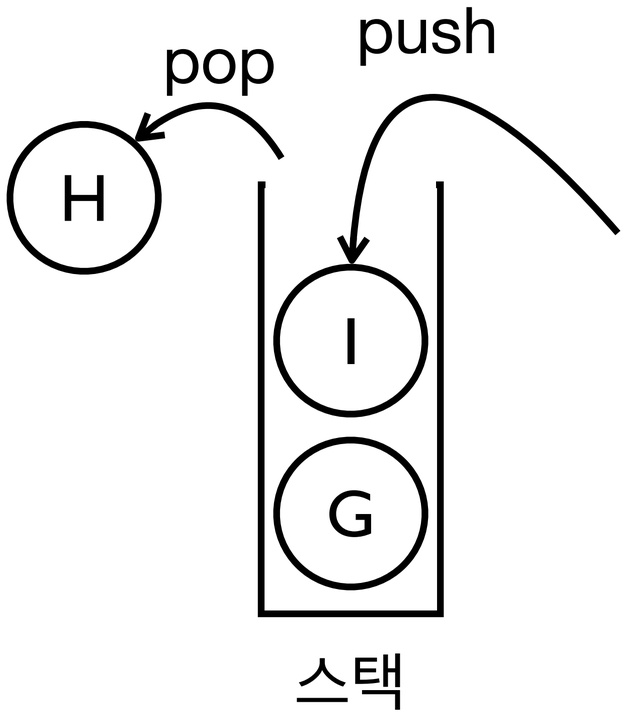
(1) 시작 노드를 스택에 push 함

(2) 스택의 top노드를 방문해 찾고자 하는 노드인지 확인함 (찾았을 경우 해당 노드를 반환 후 종료).

(3) 스택을 pop하고 pop된 노드의 인접 노드 중 방문하지 않았으면서 스택에 없는 노드들을 스택에 push함

(4) 스택이 빌 때까지 (2), (3)을 반복함



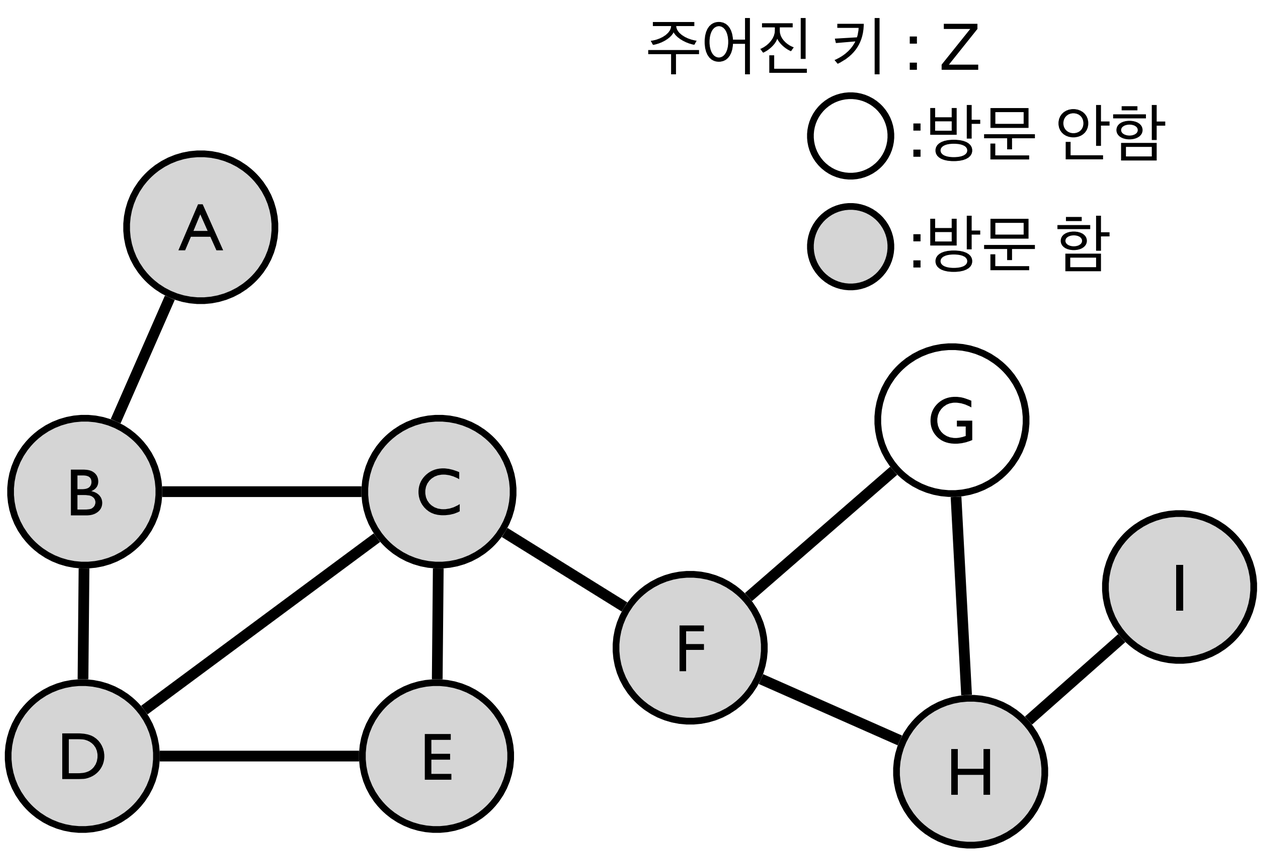


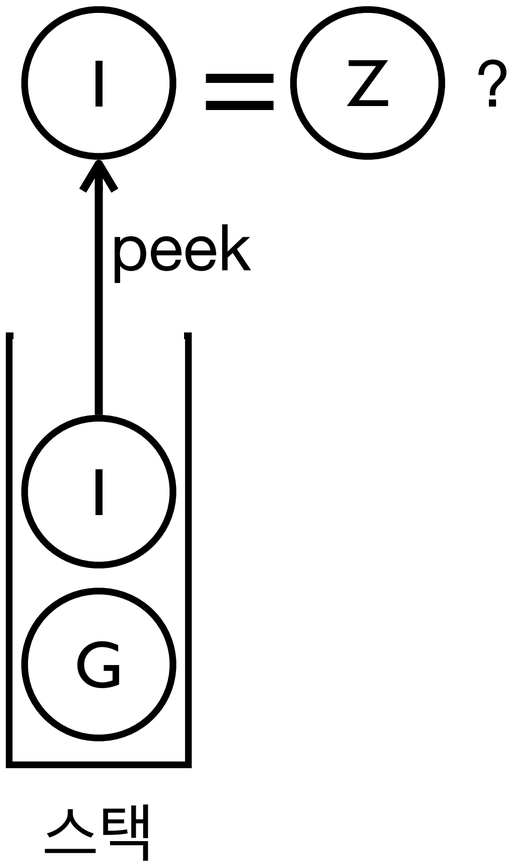
(1) 시작 노드를 스택에 push 함

(2) 스택의 top노드를 방문해 찾고자 하는 노드인지 확인함 (찾았을 경우 해당 노드를 반환 후 종료).

(3) 스택을 pop하고 pop된 노드의 인접 노드 중 방문하지 않았으면서 스택에 없는 노드들을 스택에 push함

(4) 스택이 빌 때까지 (2), (3)을 반복함



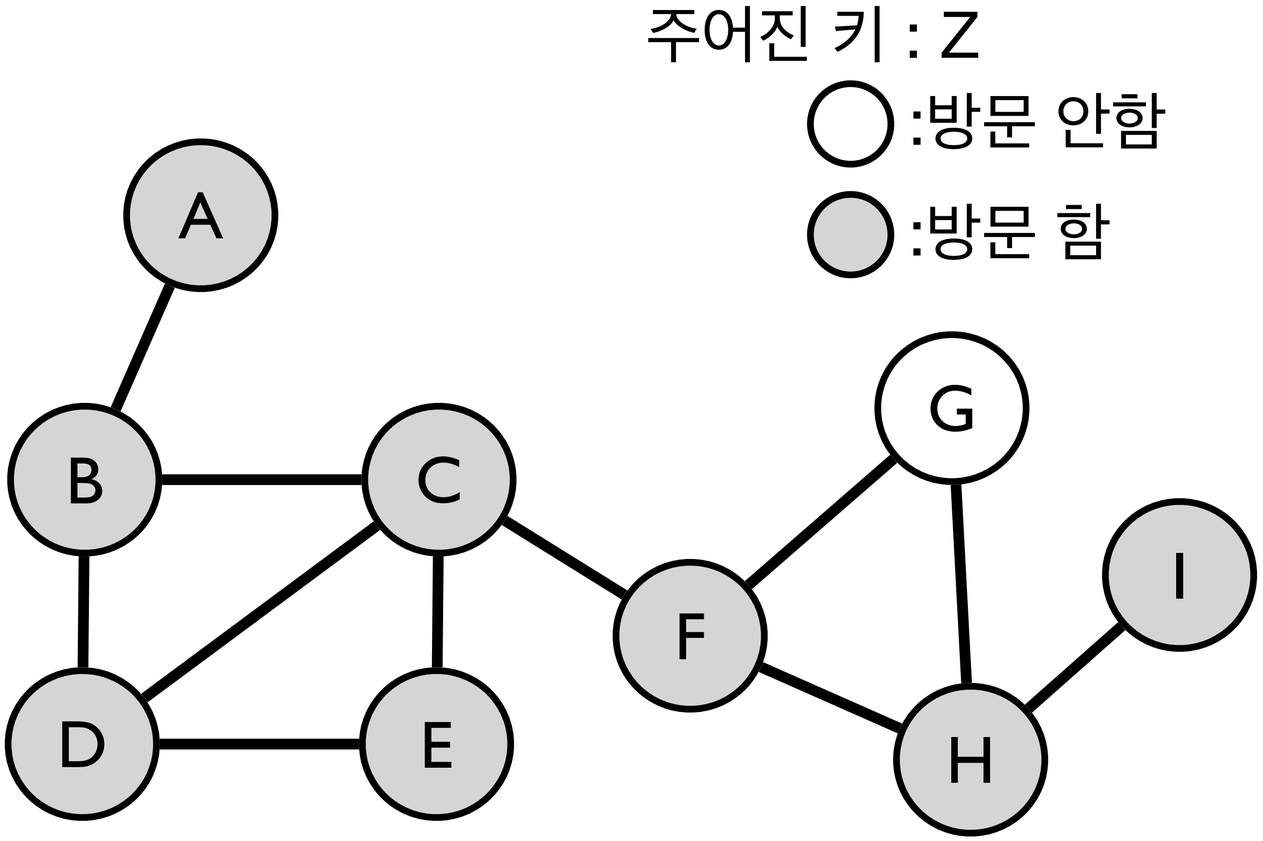


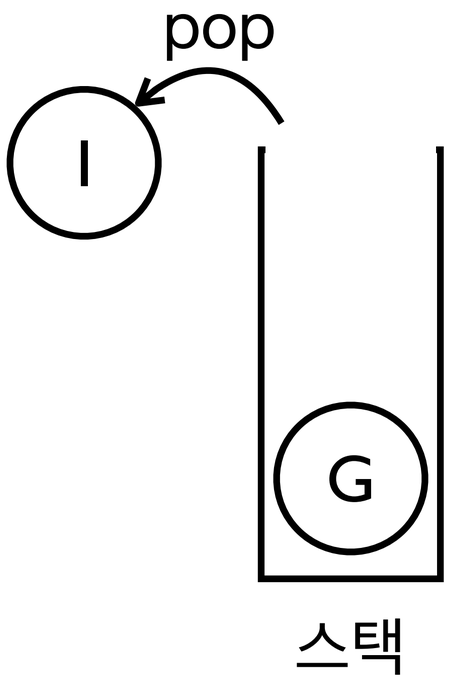
(1) 시작 노드를 스택에 push 함

(2) 스택의 top노드를 방문해 찾고자 하는 노드인지 확인함 (찾았을 경우 해당 노드를 반환 후 종료).

(3) 스택을 pop하고 pop된 노드의 인접 노드 중 방문하지 않았으면서 스택에 없는 노드들을 스택에 push함

(4) 스택이 빌 때까지 (2), (3)을 반복함



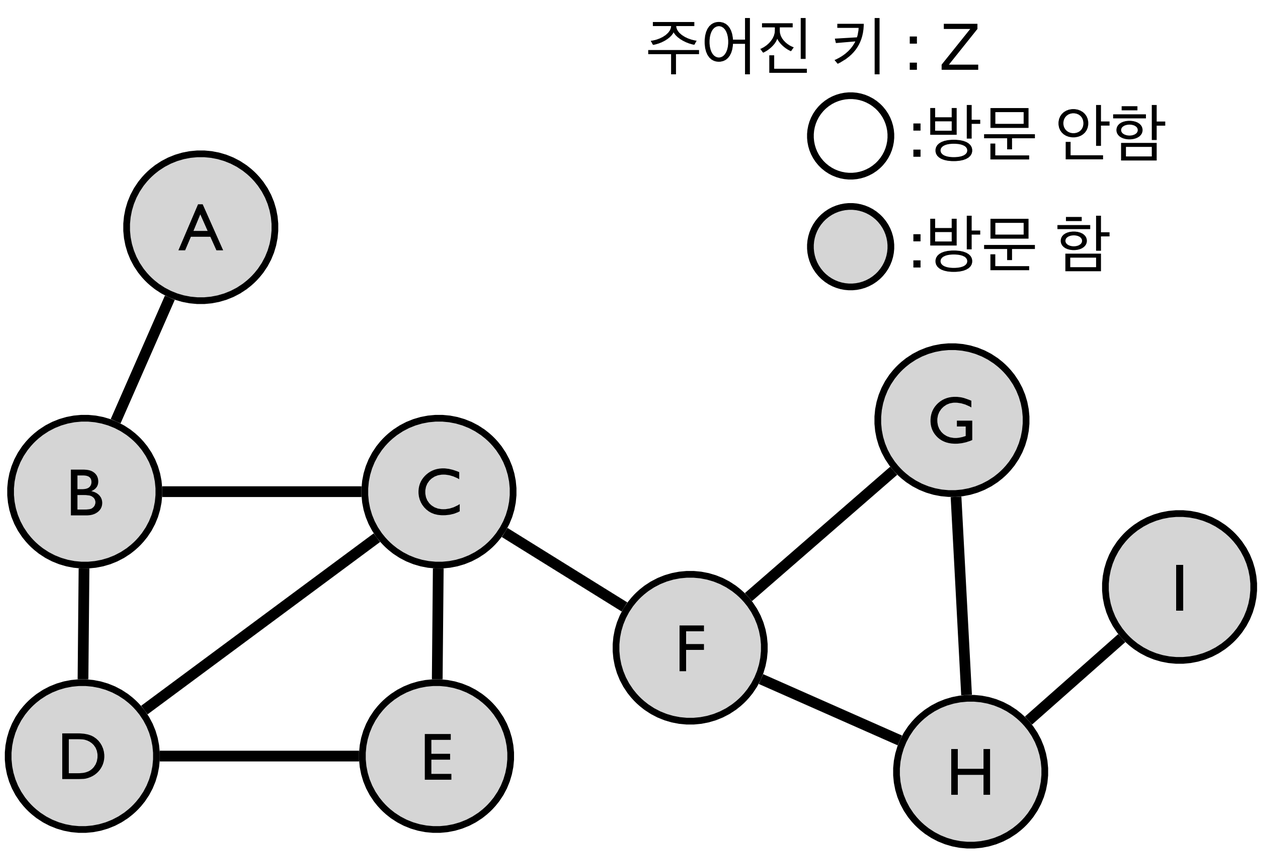


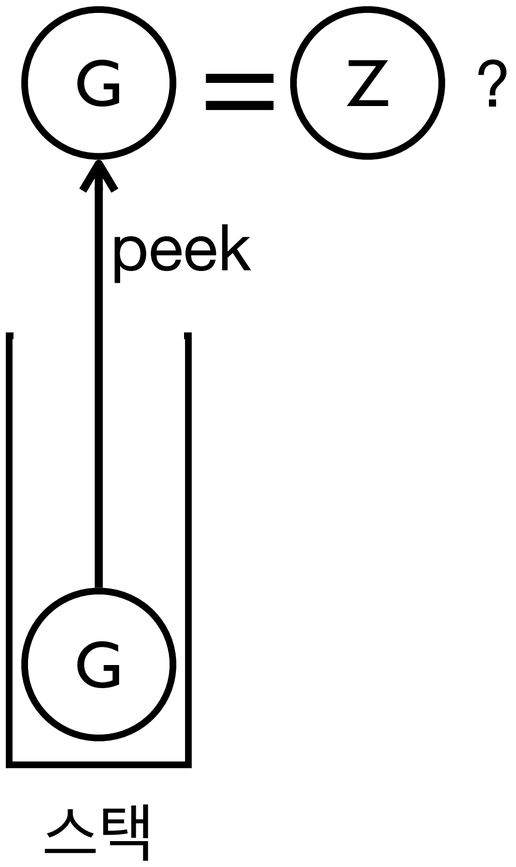
(1) 시작 노드를 스택에 push 함

(2) 스택의 top노드를 방문해 찾고자 하는 노드인지 확인함 (찾았을 경우 해당 노드를 반환 후 종료).

(3) 스택을 pop하고 pop된 노드의 인접 노드 중 방문하지 않았으면서 스택에 없는 노드들을 스택에 push함

(4) 스택이 빌 때까지 (2), (3)을 반복함



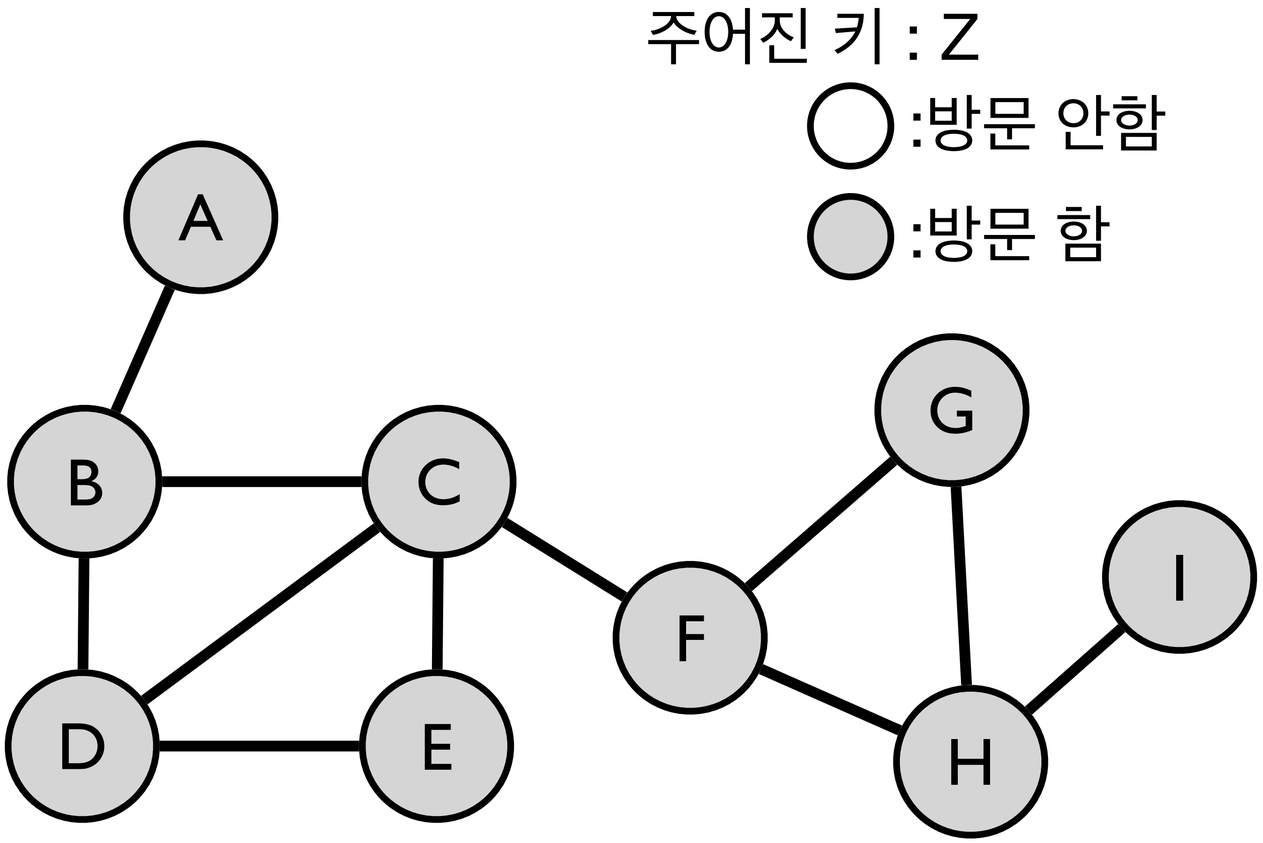


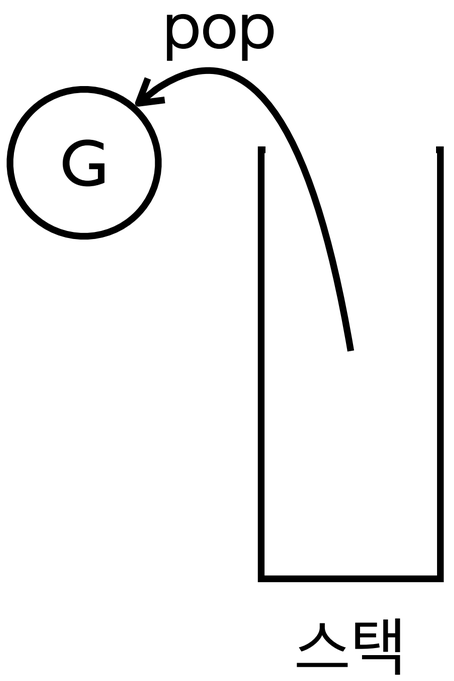
(1) 시작 노드를 스택에 push 함

(2) 스택의 top노드를 방문해 찾고자 하는 노드인지 확인함 (찾았을 경우 해당 노드를 반환 후 종료).

(3) 스택을 pop하고 pop된 노드의 인접 노드 중 방문하지 않았으면서 스택에 없는 노드들을 스택에 push함

(4) 스택이 빌 때까지 (2), (3)을 반복함



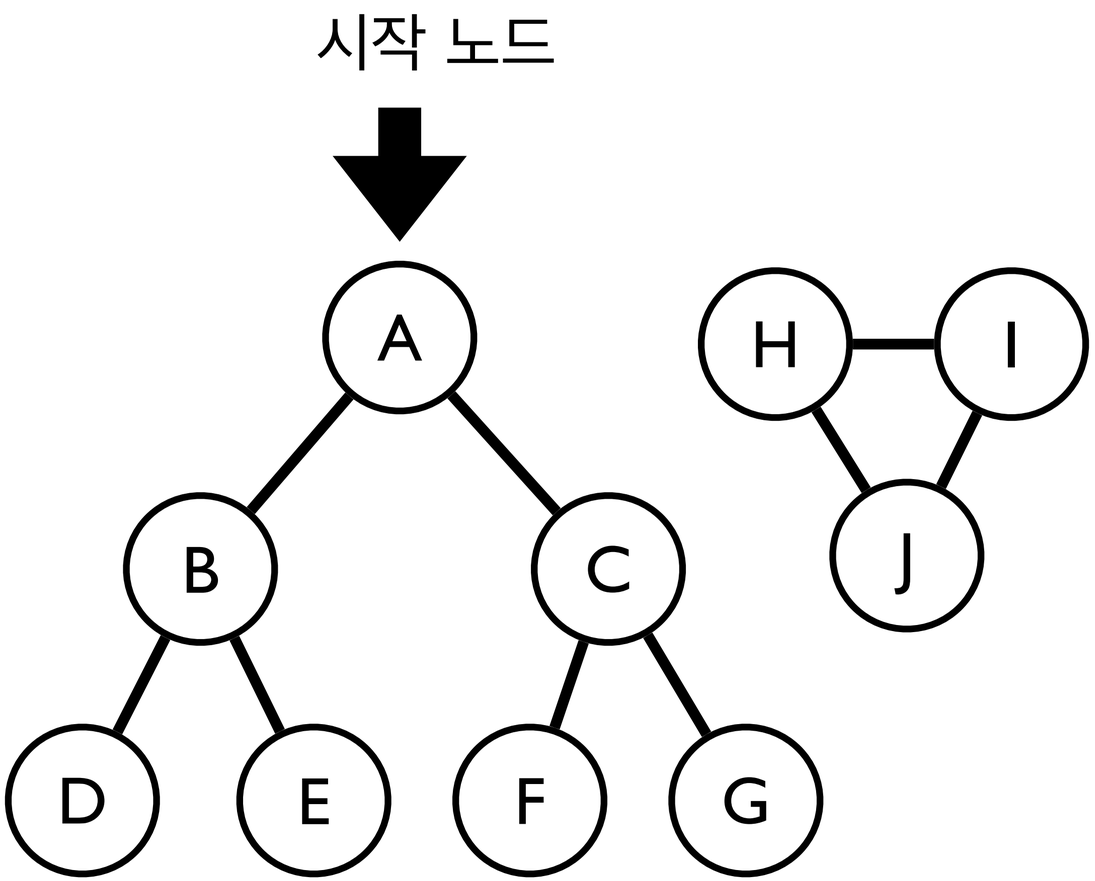


Example Code

```
#include "Stack.h"
int DFS(Graph *graph, int startNode, int targetValue) {
    if (startNode >= MAX_NODES || !(graph->nodePresent[startNode])) {
        return -1; // No such node
    }
    bool visited[MAX_NODES] = { false };
    Stack* stack = create();
    push(stack, startNode);
    while (!isEmpty(stack)) {
        int current = pop(stack);
        if (!visited[current]) {
            visited[current] = true;
            if (graph->data[current] == targetValue) {
                return current;
            }
            for (int i = MAX_NODES - 1; i >= 0; i--) {
                if (graph->adjacencyMatrix[current][i] && graph->nodePresent[i] && !(visited[i])) {
                    push(stack, i);
                }
            }
        }
    }
    return -1; // No such node
}
```

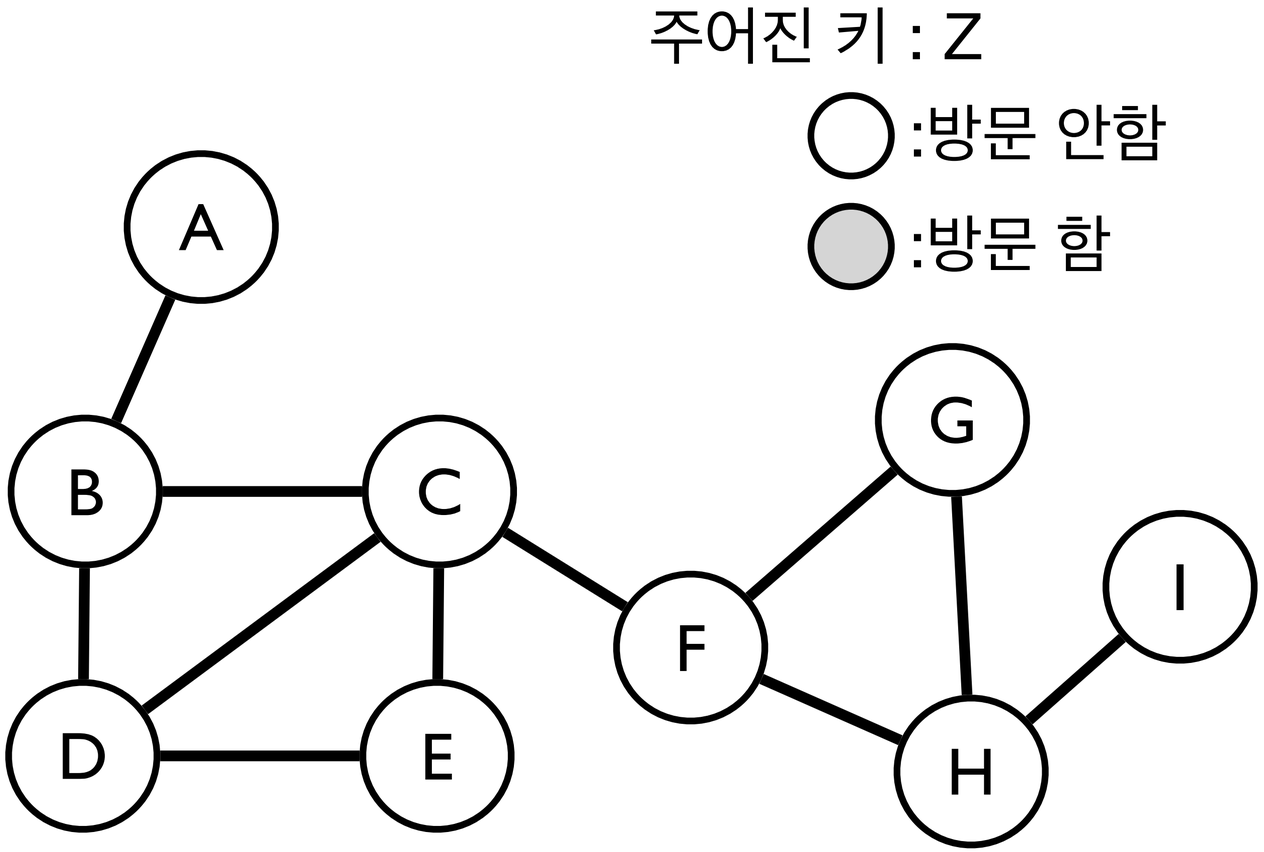
Example

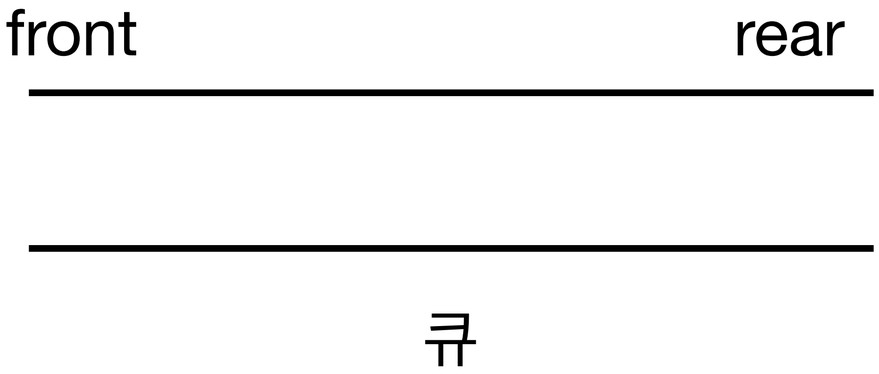
- 왼쪽의 노드가 10개인 그래프 데이터에서 DFS순회를 하였을 때 방문 노드의 순서를 기술하시오



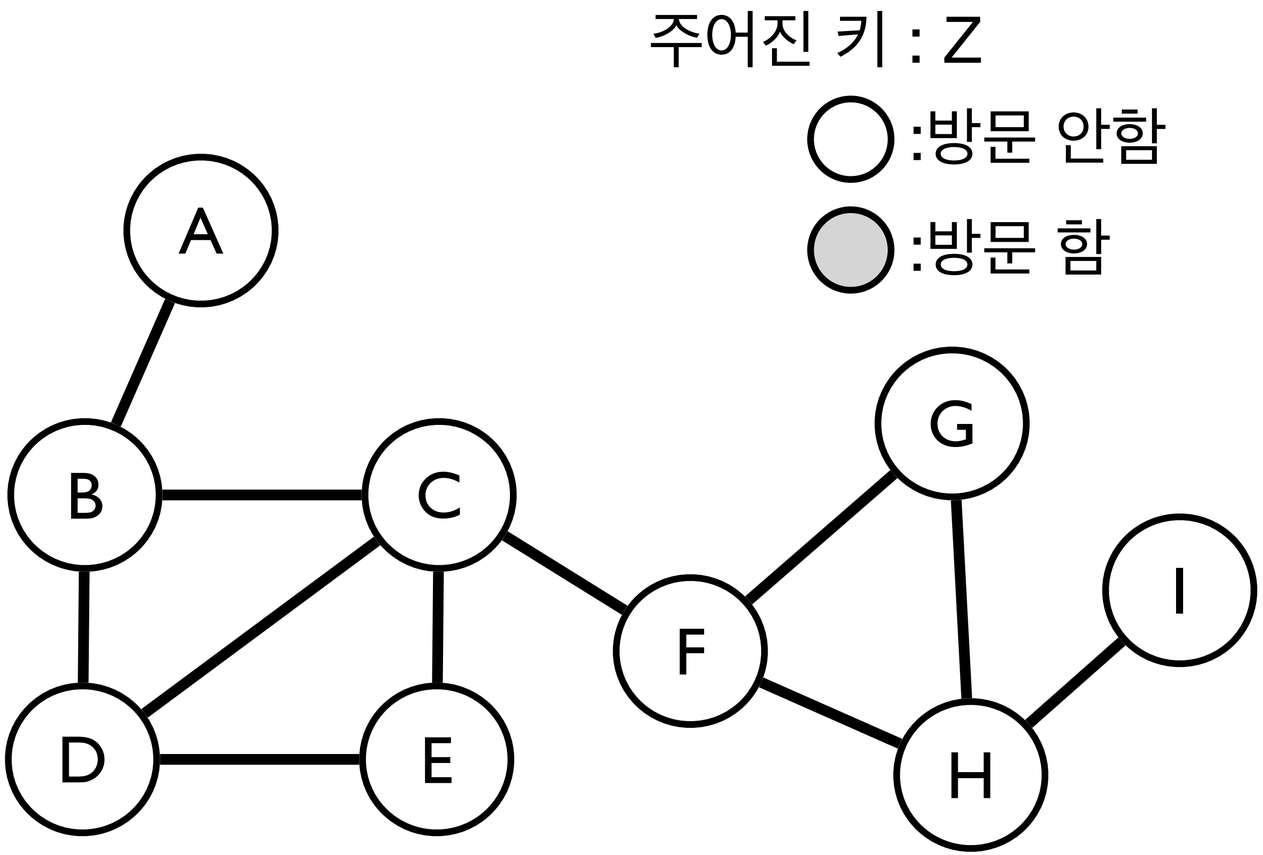
Breadth First Search (넓이 우선 탐색)

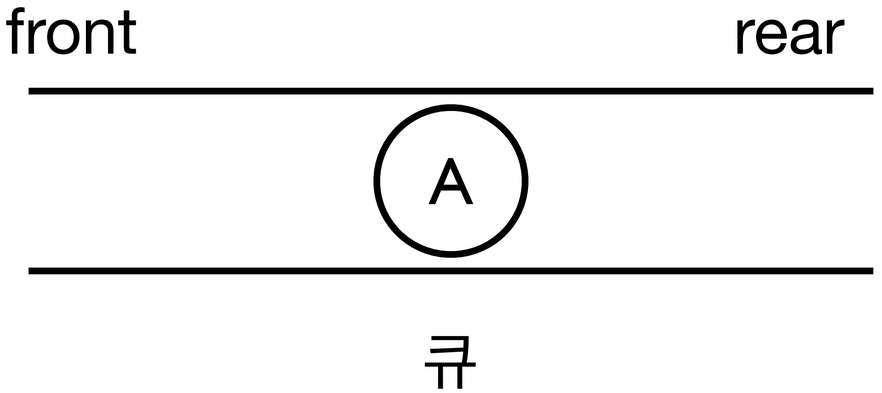
- 넓이 우선 탐색(DFS)은 가까운 노드들부터 먼 노드들까지 순차적으로 탐색함





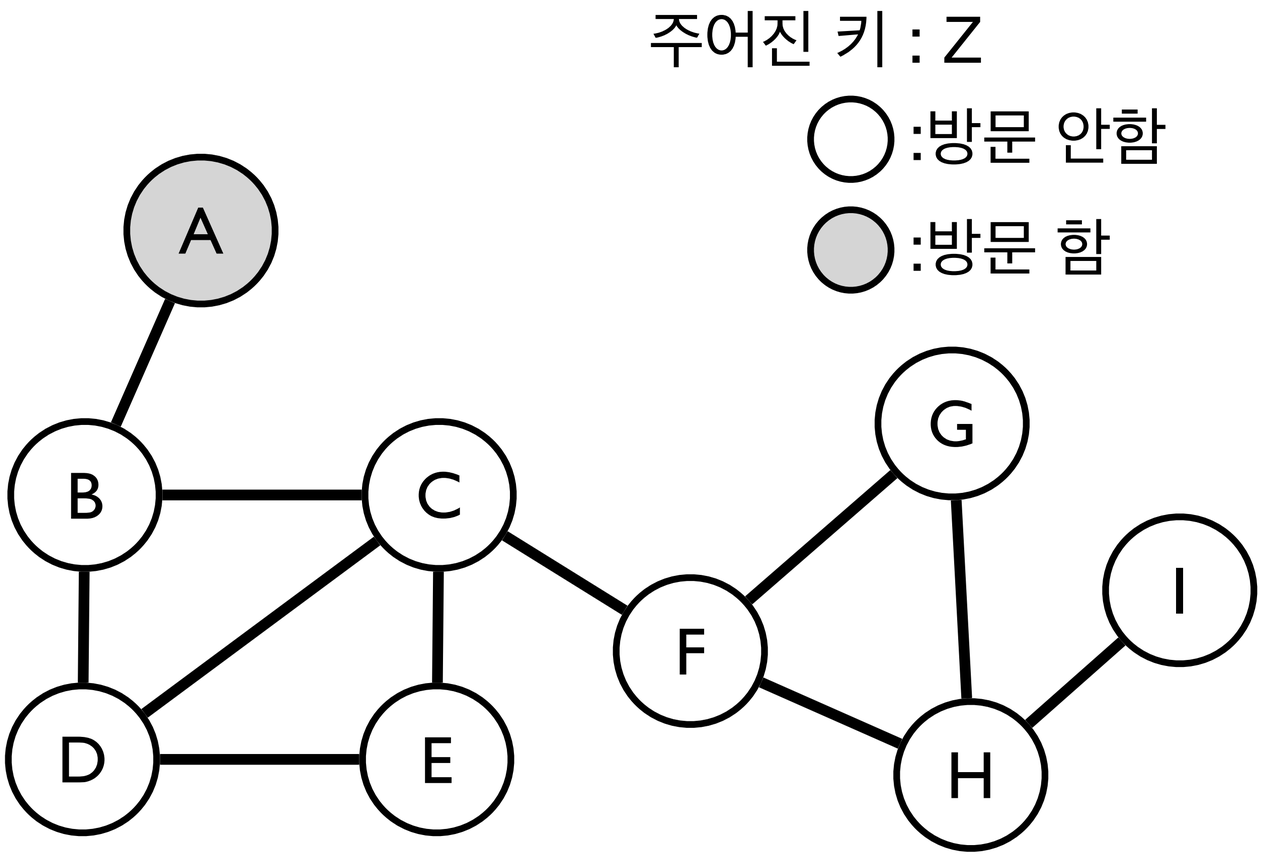
(1) 시작 노드를 큐에 enqueue 함

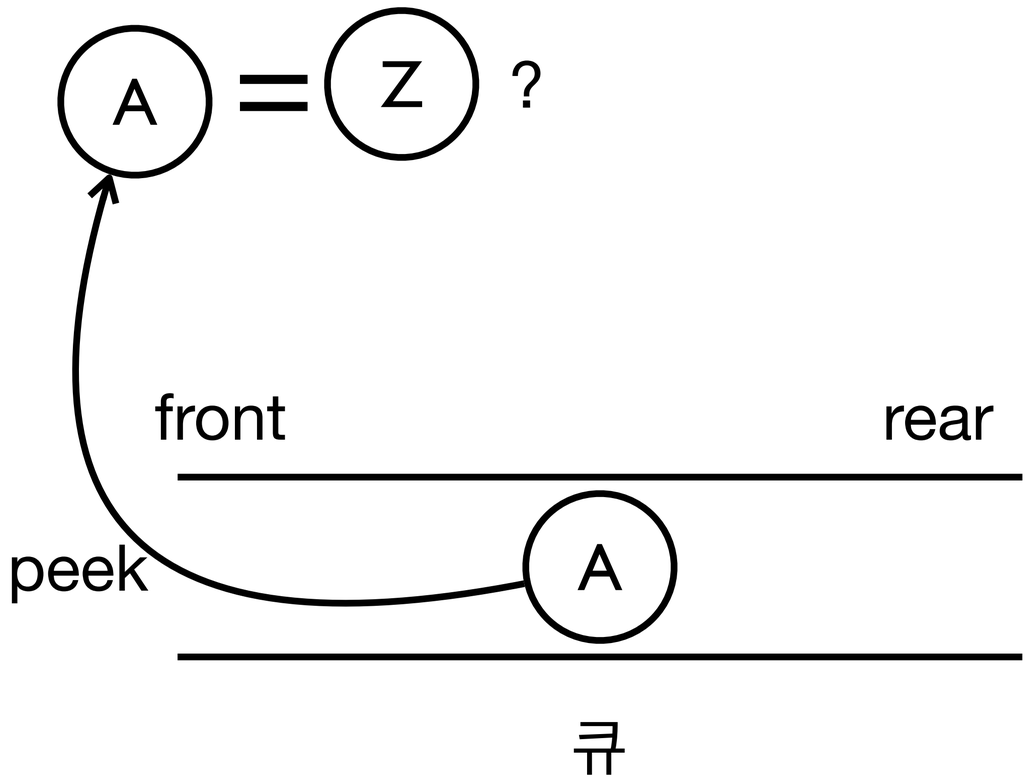




(1) 시작 노드를 큐에 enqueue 함

(2) 큐의 front의 노드가 찾고자 하는 노드인지 확인함 (찾았을 경우 해당 노드를 반환 후 종료).



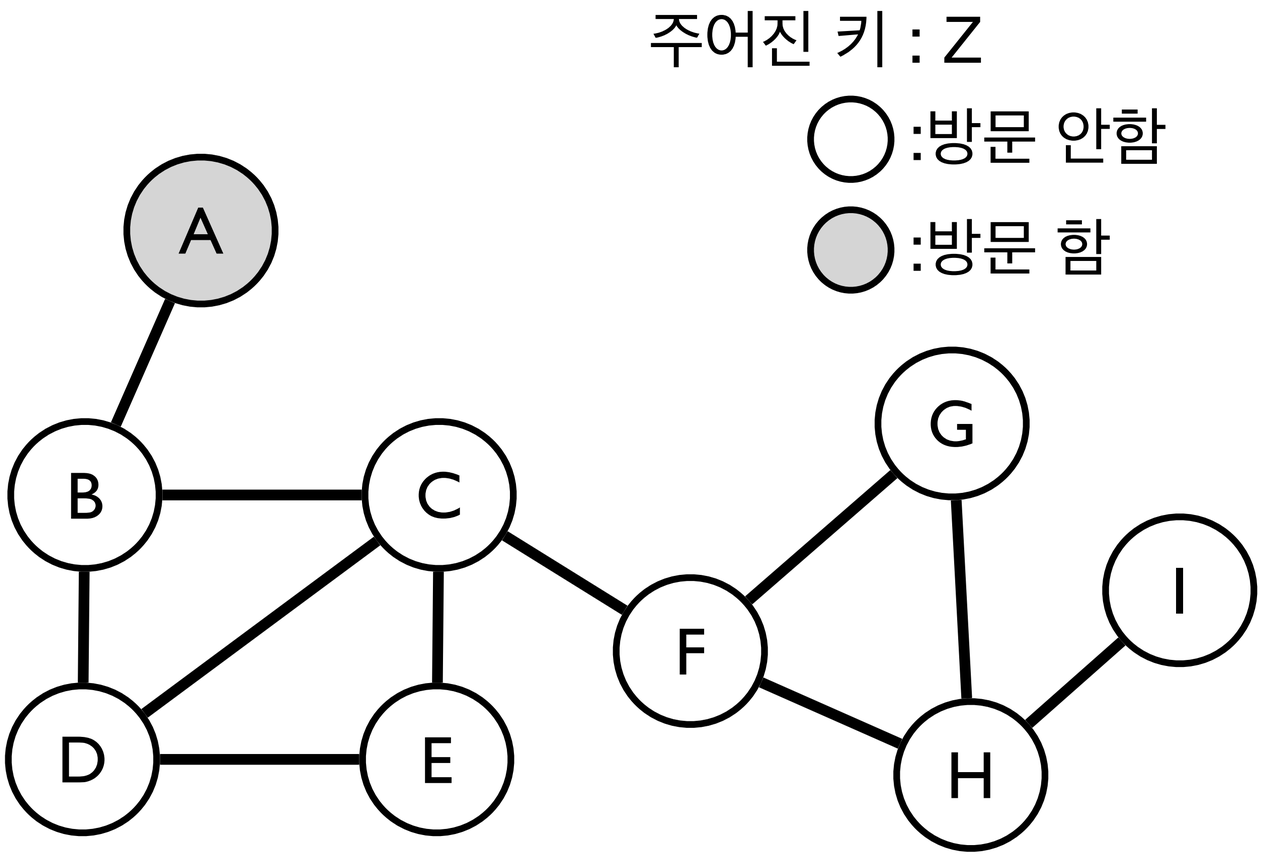


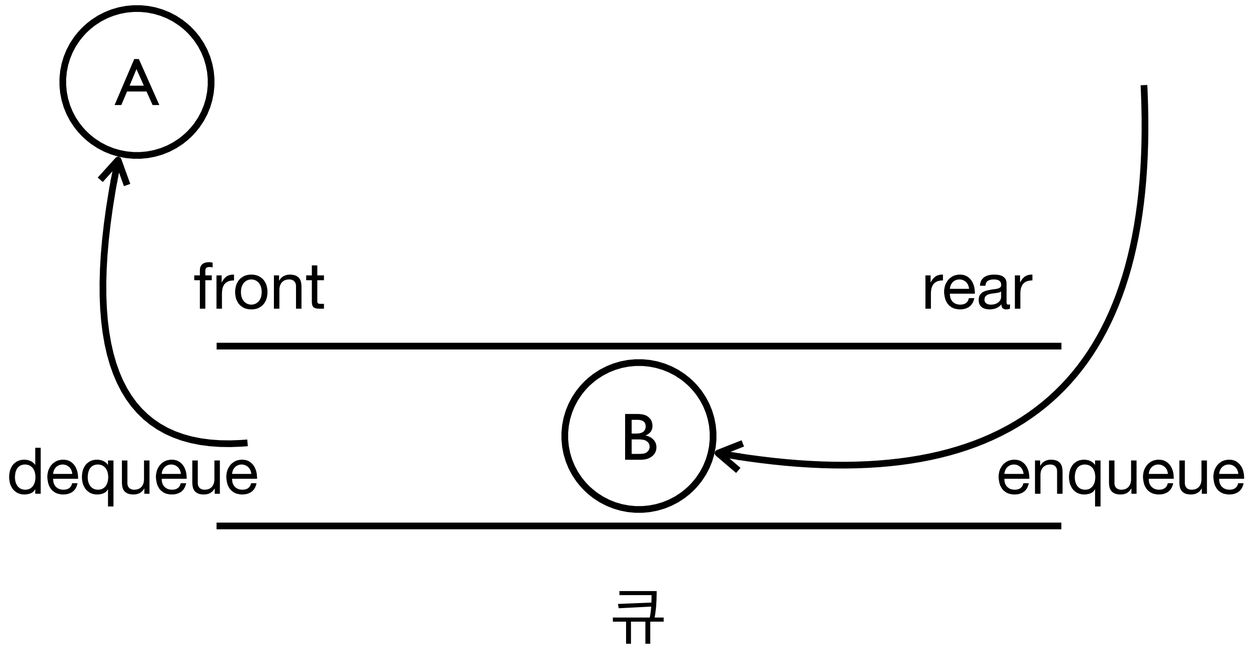
(1) 시작 노드를 큐에 enqueue 함

(2) 큐의 front의 노드가 찾고자 하는 노드인지 확인함 (찾았을 경우 해당 노드를 반환 후 종료).

(3) dequeue 후 dequeue한 노드의 인접 노드들 중 방문하지 않았으면서 큐에 없는 노드들을 큐에 삽입(enqueue)함

(4) 큐가 빌 때까지 (2), (3)을 반복함



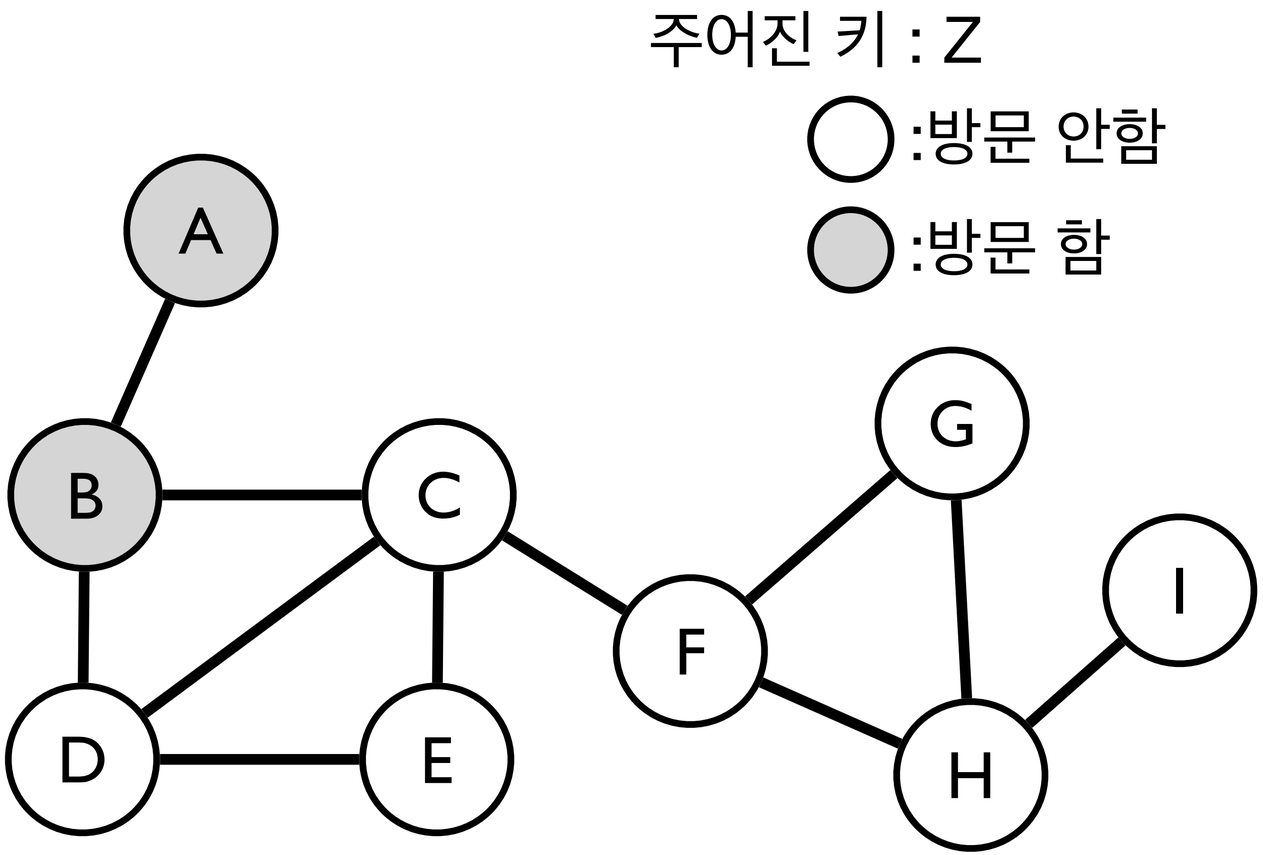


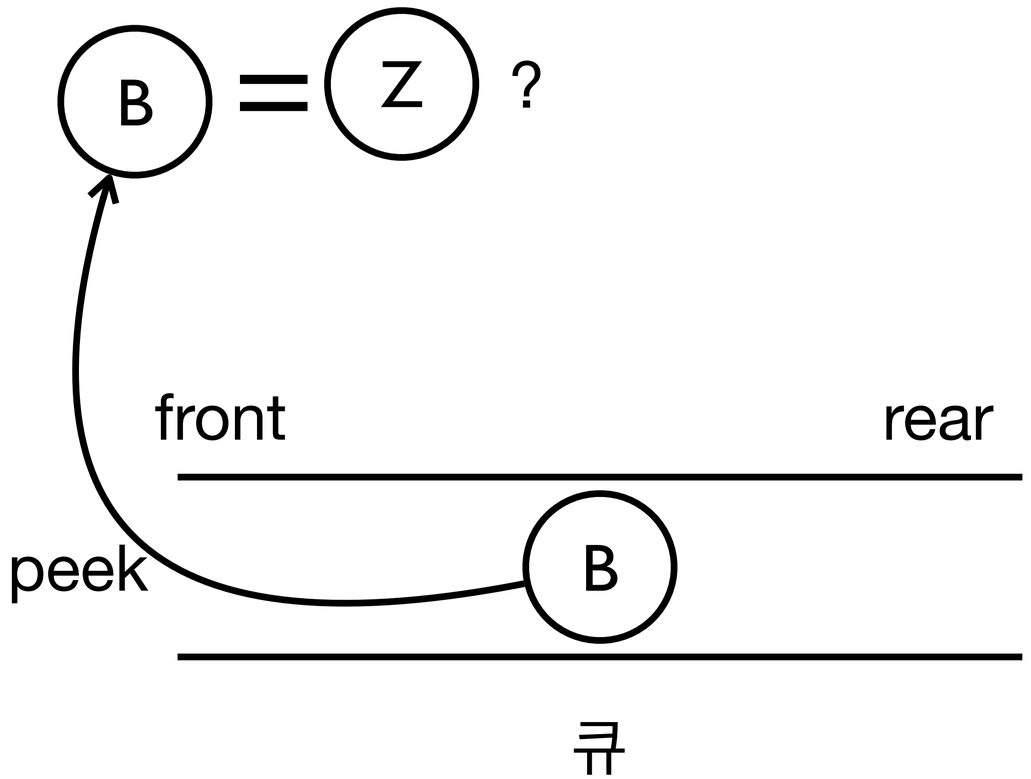
(1) 시작 노드를 큐에 enqueue 함

(2) 큐의 front의 노드가 찾고자 하는 노드인지 확인함 (찾았을 경우 해당 노드를 반환 후 종료).

(3) dequeue 후 dequeue한 노드의 인접 노드들 중 방문하지 않았으면서 큐에 없는 노드들을 큐에 삽입(enqueue)함

(4) 큐가 빌 때까지 (2), (3)을 반복함



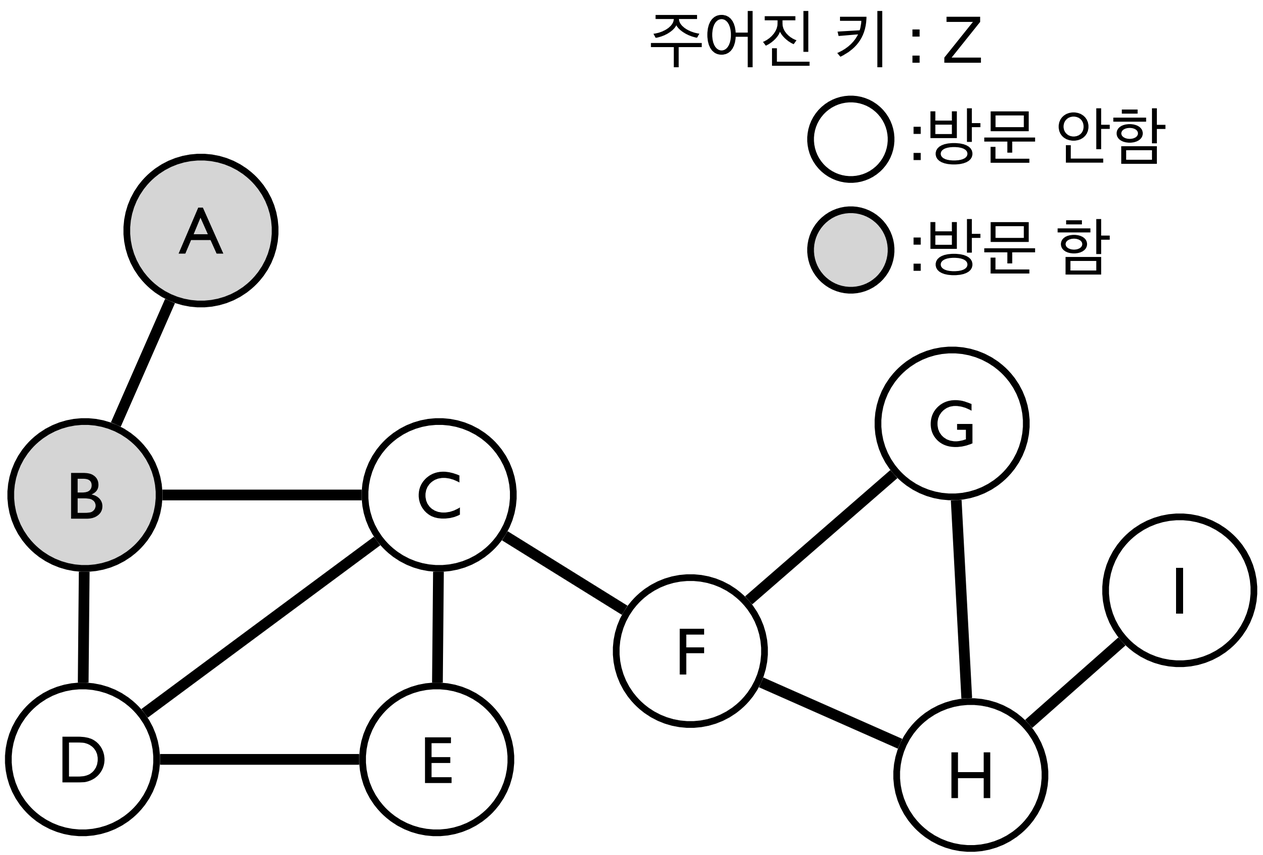


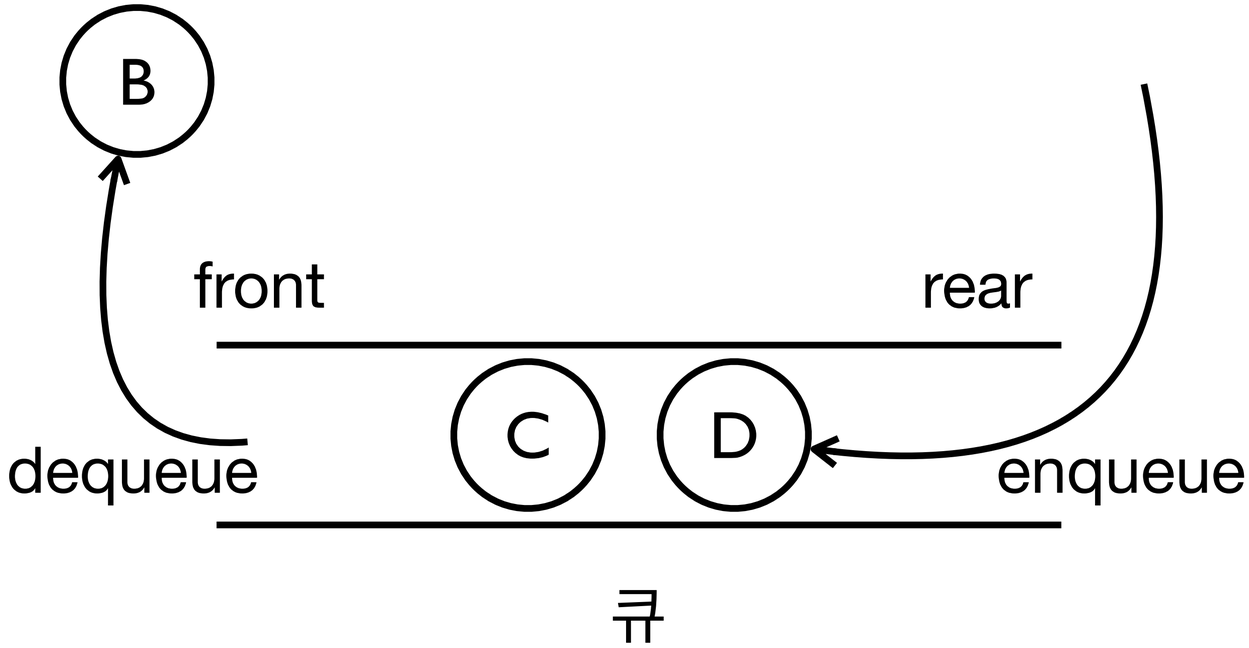
(1) 시작 노드를 큐에 enqueue 함

(2) 큐의 front의 노드가 찾고자 하는 노드인지 확인함 (찾았을 경우 해당 노드를 반환 후 종료).

(3) dequeue 후 dequeue한 노드의 인접 노드들 중 방문하지 않았으면서 큐에 없는 노드들을 큐에 삽입(enqueue)함

(4) 큐가 빌 때까지 (2), (3)을 반복함



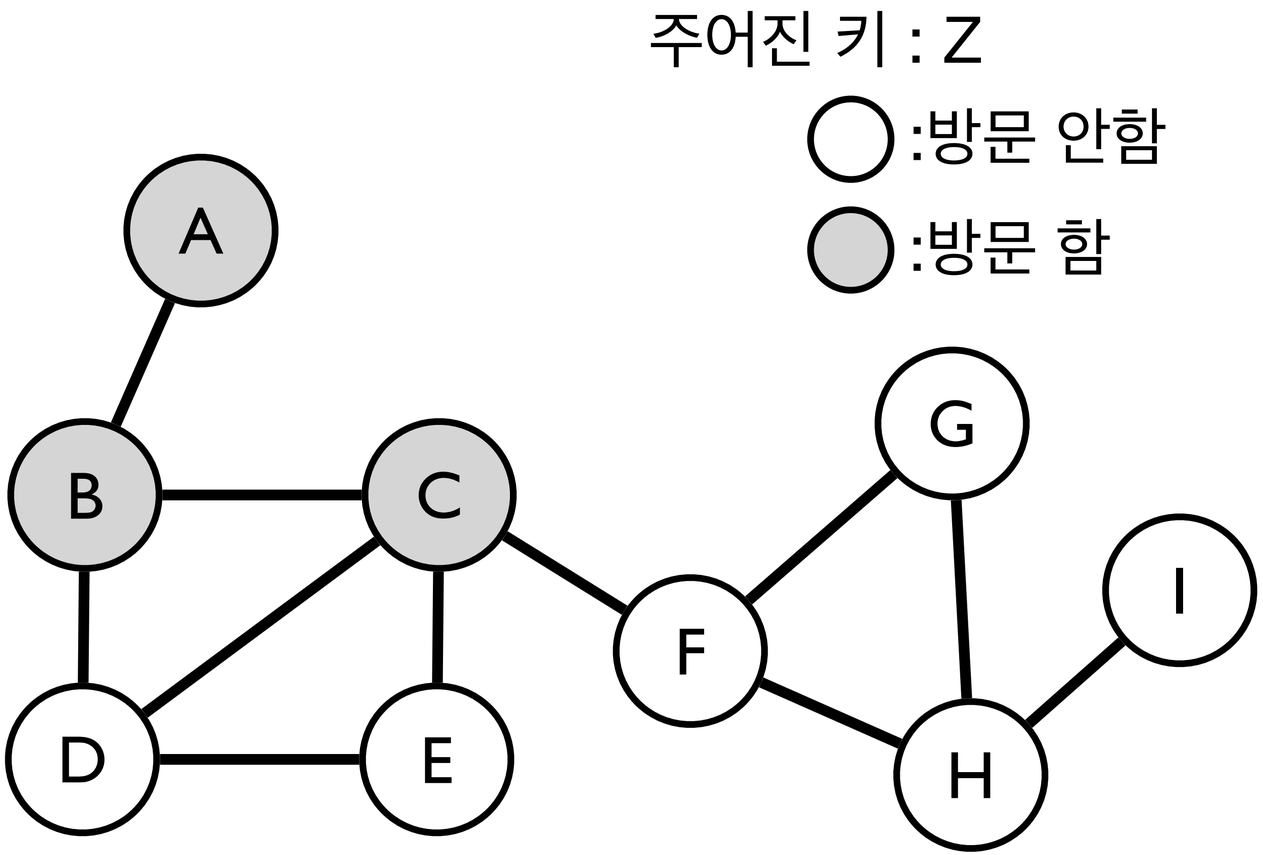


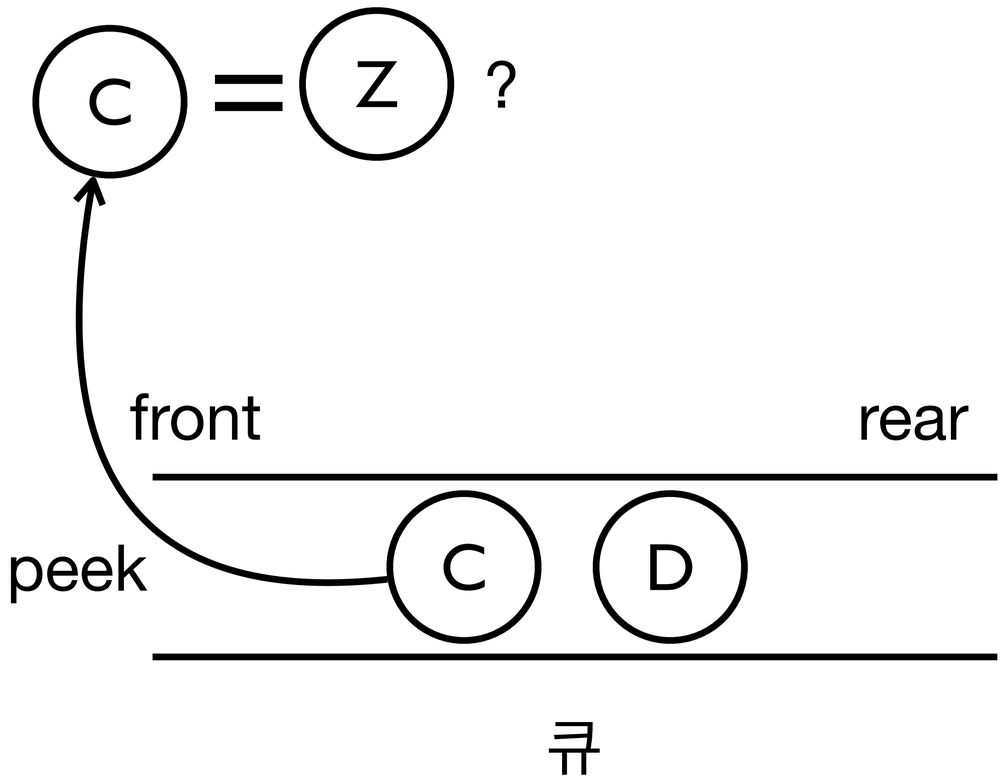
(1) 시작 노드를 큐에 enqueue 함

(2) 큐의 front의 노드가 찾고자 하는 노드인지 확인함 (찾았을 경우 해당 노드를 반환 후 종료).

(3) dequeue 후 dequeue한 노드의 인접 노드들 중 방문하지 않았으면서 큐에 없는 노드들을 큐에 삽입(enqueue)함

(4) 큐가 빌 때까지 (2), (3)을 반복함



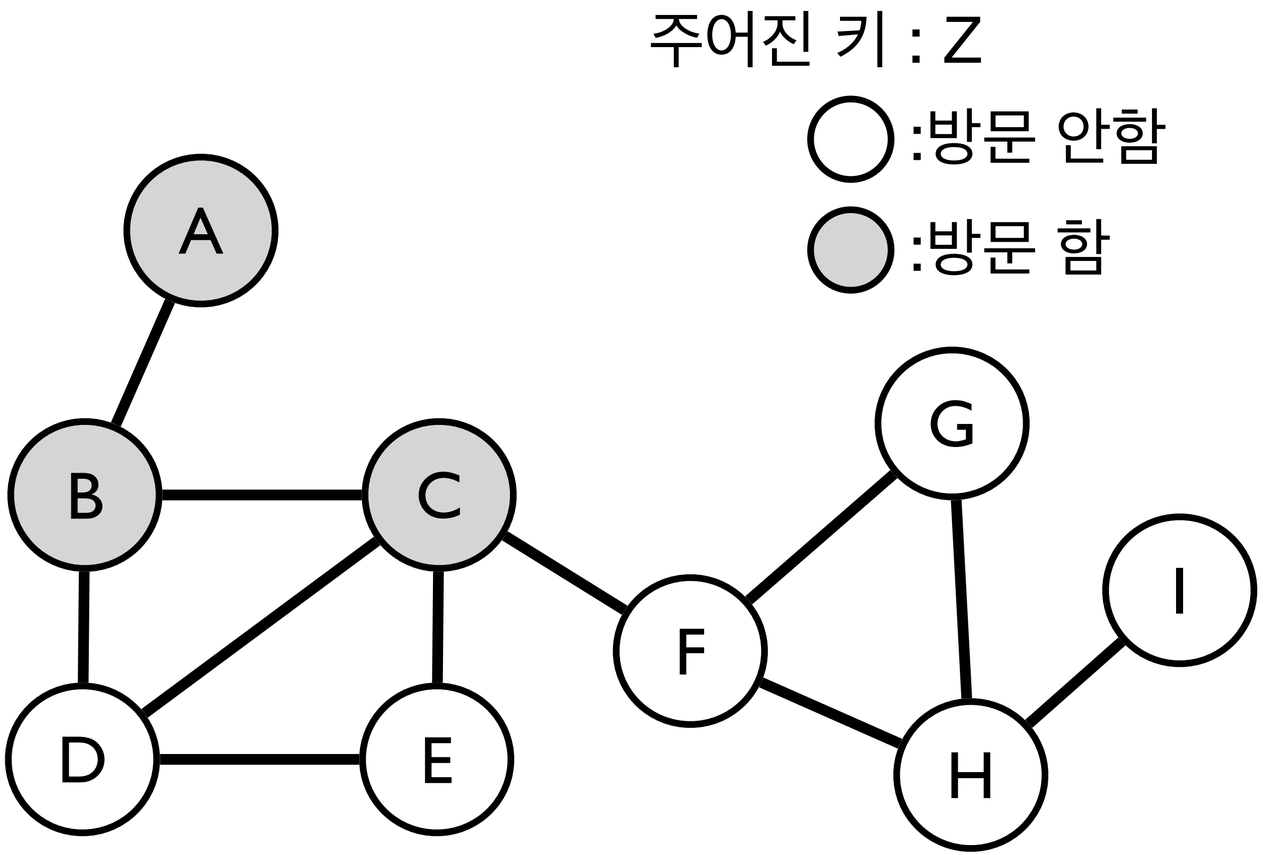


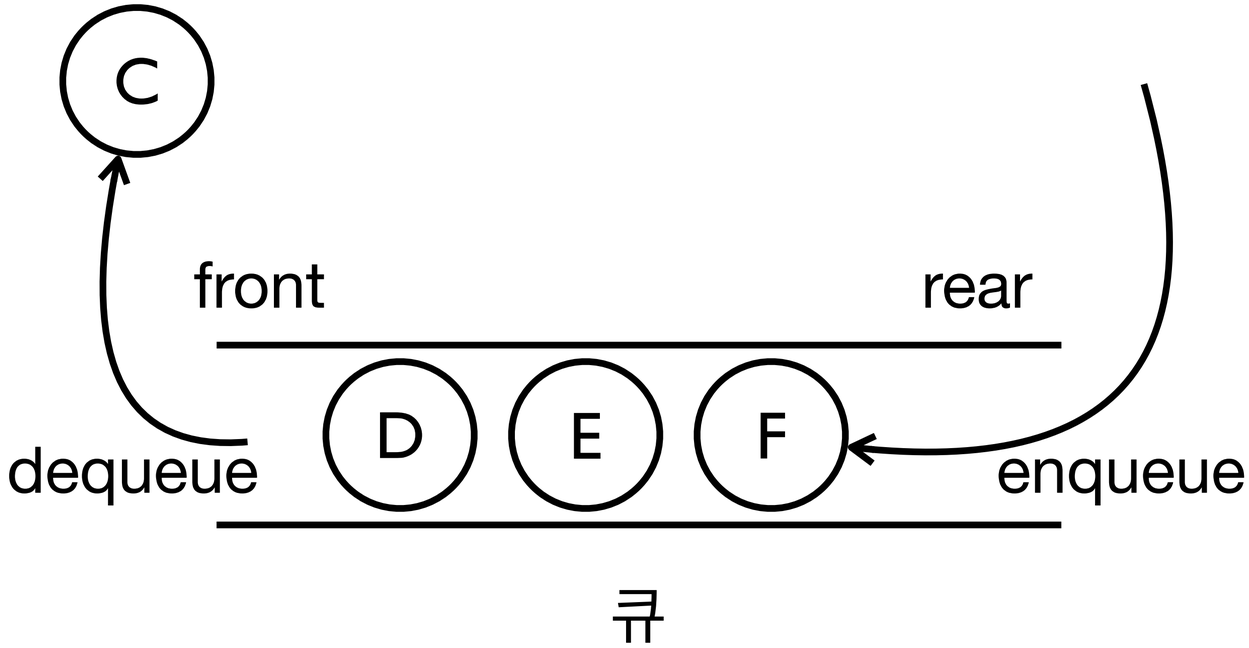
(1) 시작 노드를 큐에 enqueue 함

(2) 큐의 front의 노드가 찾고자 하는 노드인지 확인함 (찾았을 경우 해당 노드를 반환 후 종료).

(3) dequeue 후 dequeue한 노드의 인접 노드들 중 방문하지 않았으면서 큐에 없는 노드들을 큐에 삽입(enqueue)함

(4) 큐가 빌 때까지 (2), (3)을 반복함



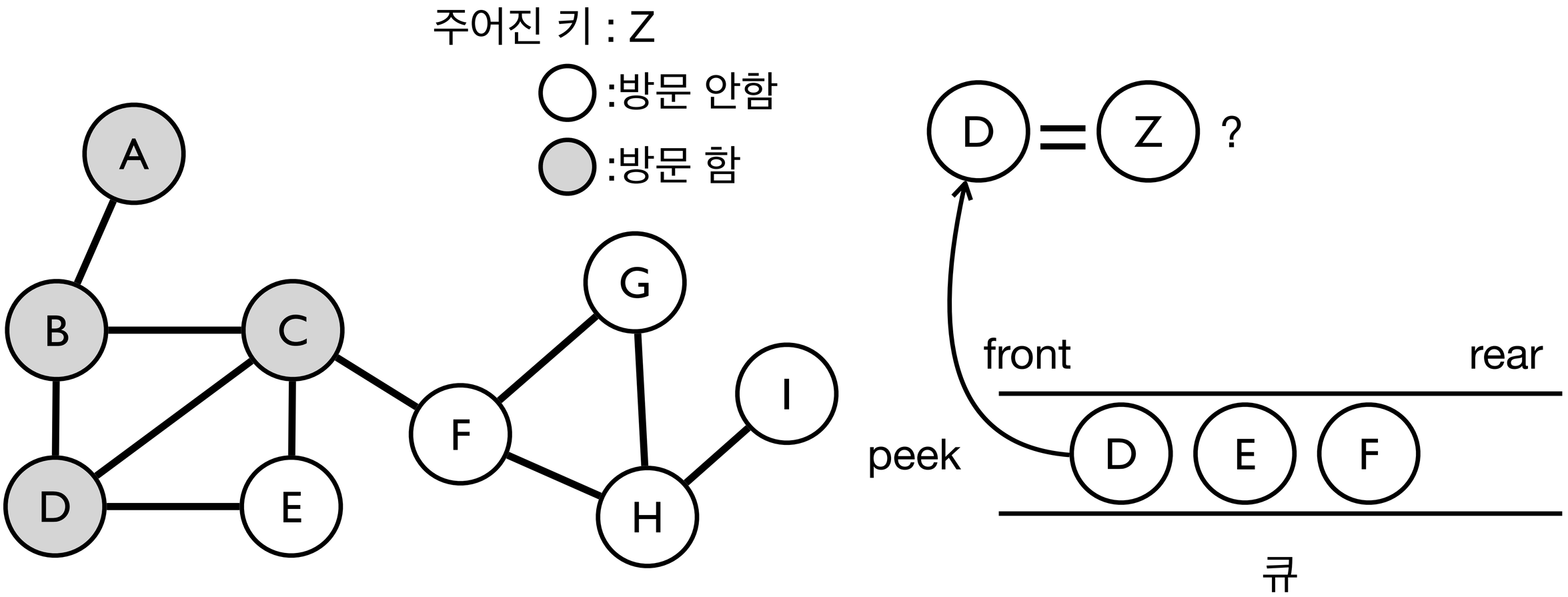


(1) 시작 노드를 큐에 enqueue 함

(2) 큐의 front의 노드가 찾고자 하는 노드인지 확인함 (찾았을 경우 해당 노드를 반환 후 종료).

(3) dequeue 후 dequeue한 노드의 인접 노드들 중 방문하지 않았으면서 큐에 없는 노드들을 큐에 삽입(enqueue)함

(4) 큐가 빌 때까지 (2), (3)을 반복함

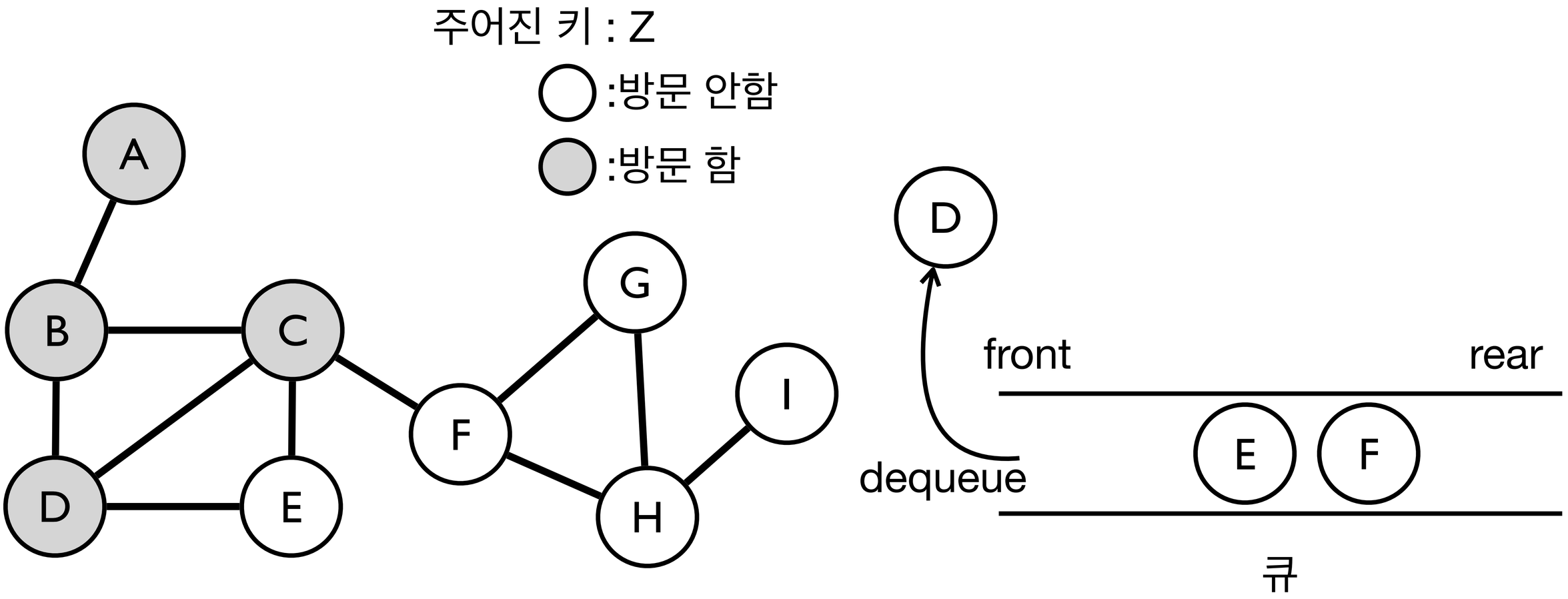


(1) 시작 노드를 큐에 enqueue 함

(2) 큐의 front의 노드가 찾고자 하는 노드인지 확인함 (찾았을 경우 해당 노드를 반환 후 종료).

(3) dequeue 후 dequeue한 노드의 인접 노드들 중 방문하지 않았으면서 큐에 없는 노드들을 큐에 삽입(enqueue)함

(4) 큐가 빌 때까지 (2), (3)을 반복함

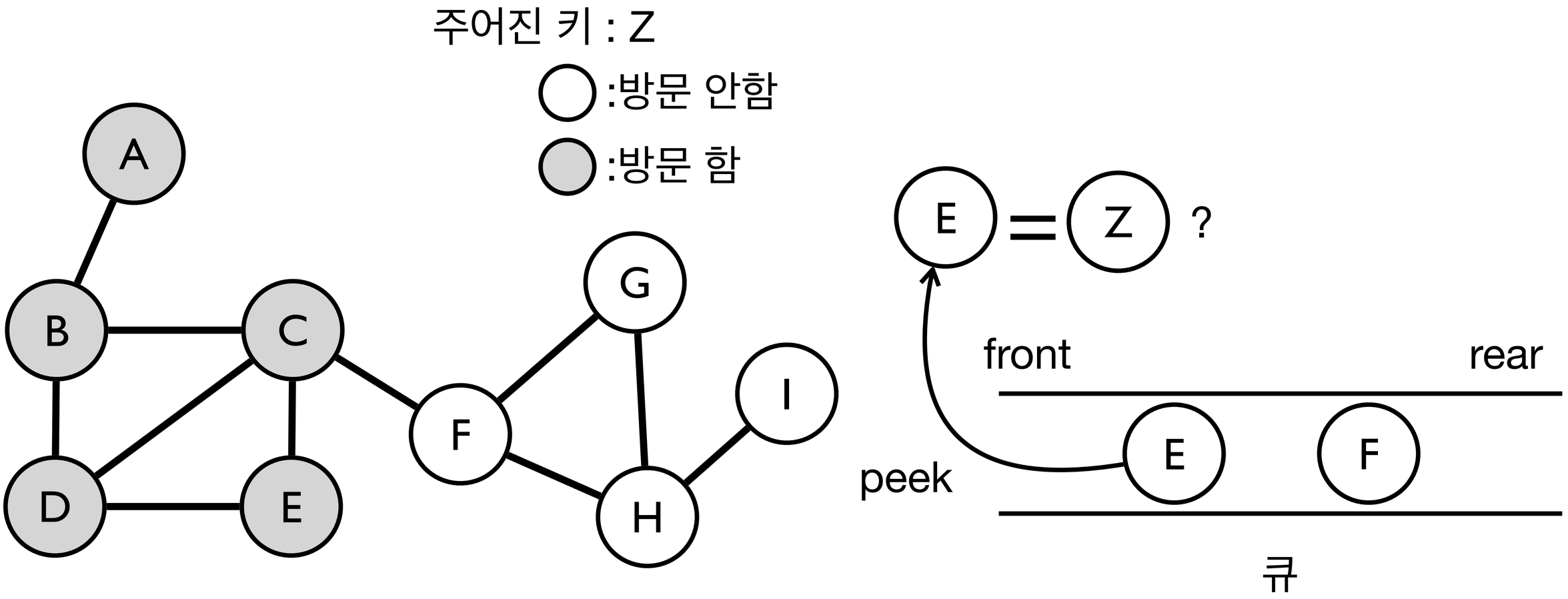


(1) 시작 노드를 큐에 enqueue 함

(2) 큐의 front의 노드가 찾고자 하는 노드인지 확인함 (찾았을 경우 해당 노드를 반환 후 종료).

(3) dequeue 후 dequeue한 노드의 인접 노드들 중 방문하지 않았으면서 큐에 없는 노드들을 큐에 삽입(enqueue)함

(4) 큐가 빌 때까지 (2), (3)을 반복함

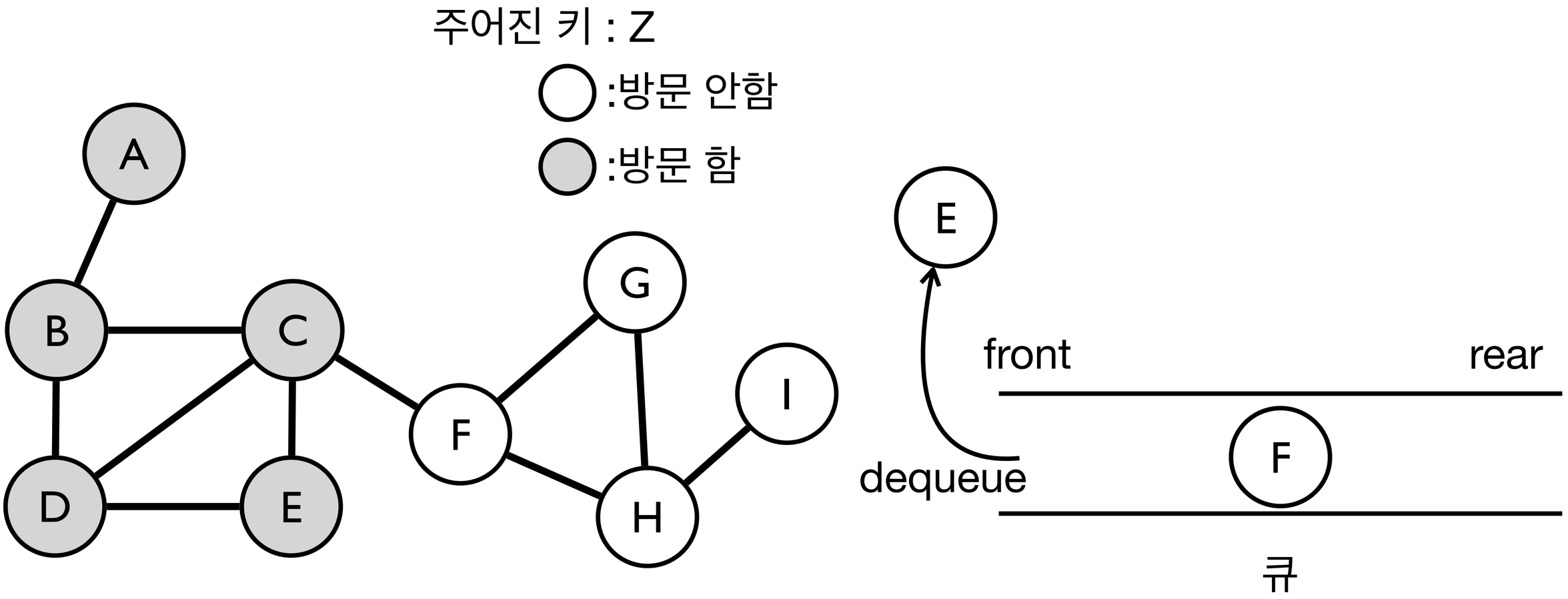


(1) 시작 노드를 큐에 enqueue 함

(2) 큐의 front의 노드가 찾고자 하는 노드인지 확인함 (찾았을 경우 해당 노드를 반환 후 종료).

(3) dequeue 후 dequeue한 노드의 인접 노드들 중 방문하지 않았으면서 큐에 없는 노드들을 큐에 삽입(enqueue)함

(4) 큐가 빌 때까지 (2), (3)을 반복함

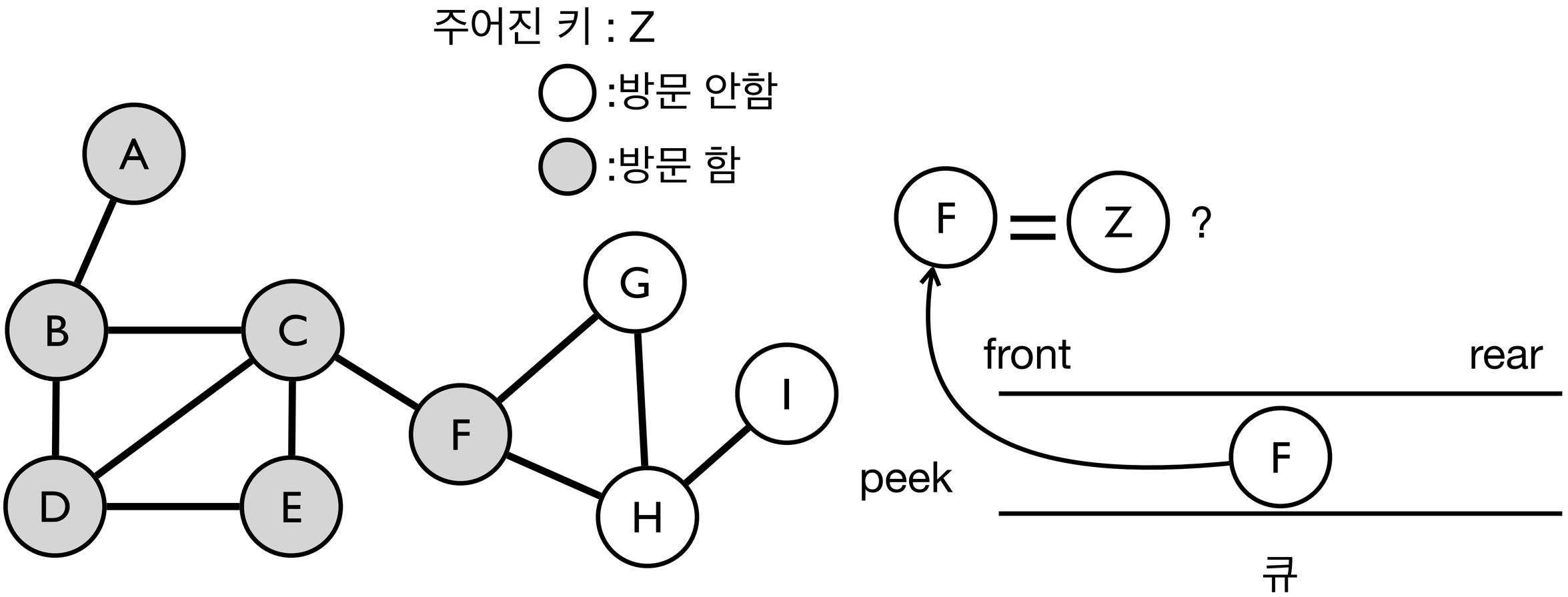


(1) 시작 노드를 큐에 enqueue 함

(2) 큐의 front의 노드가 찾고자 하는 노드인지 확인함 (찾았을 경우 해당 노드를 반환 후 종료).

(3) dequeue 후 dequeue한 노드의 인접 노드들 중 방문하지 않았으면서 큐에 없는 노드들을 큐에 삽입(enqueue)함

(4) 큐가 빌 때까지 (2), (3)을 반복함

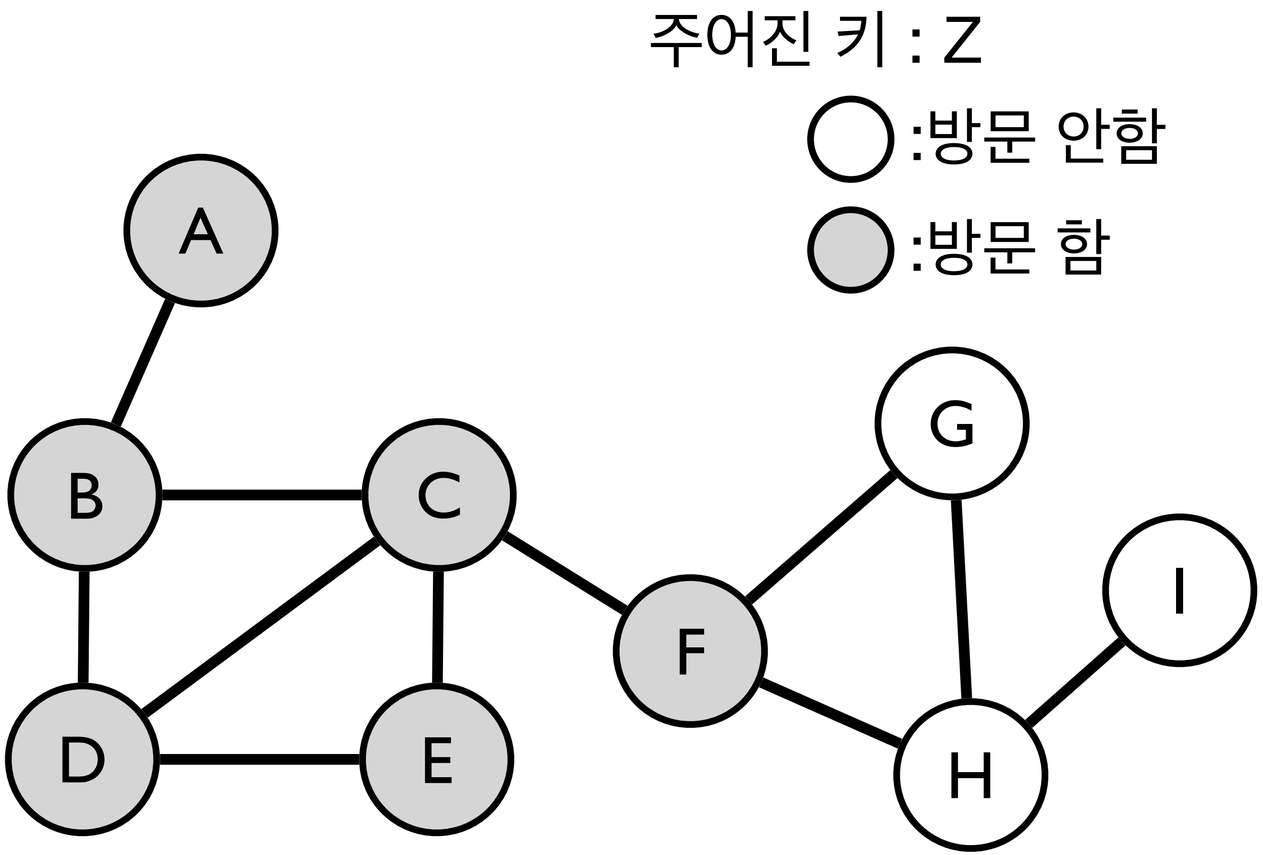


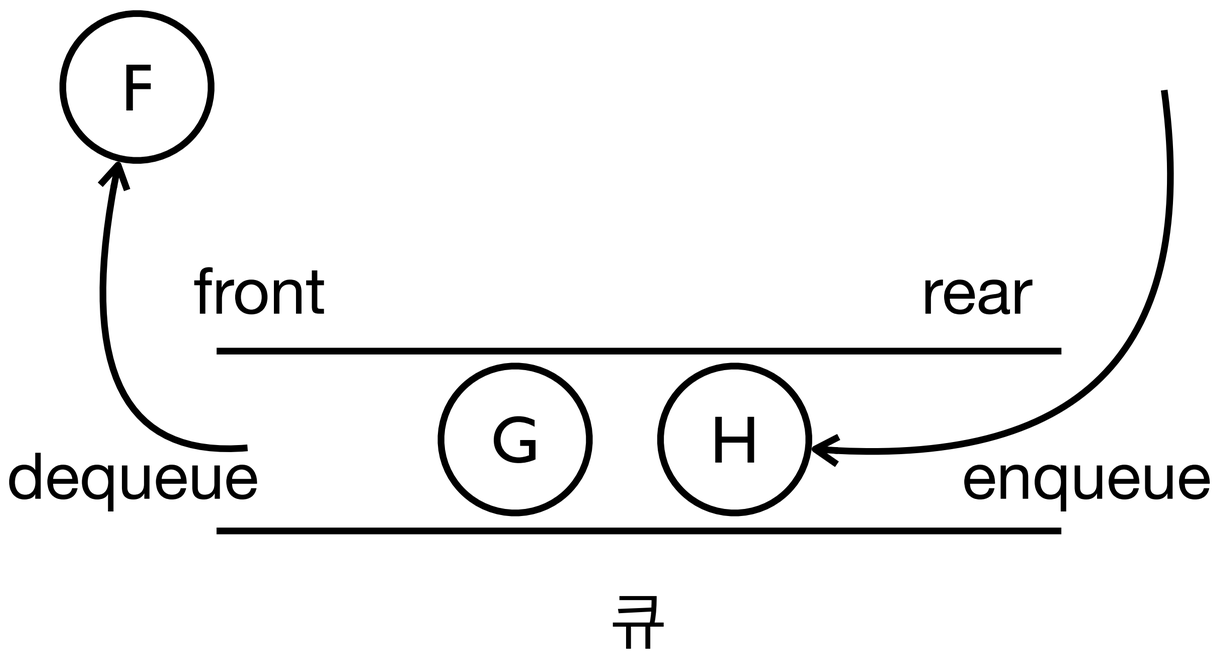
(1) 시작 노드를 큐에 enqueue 함

(2) 큐의 front의 노드가 찾고자 하는 노드인지 확인함 (찾았을 경우 해당 노드를 반환 후 종료).

(3) dequeue 후 dequeue한 노드의 인접 노드들 중 방문하지 않았으면서 큐에 없는 노드들을 큐에 삽입(enqueue)함

(4) 큐가 빌 때까지 (2), (3)을 반복함



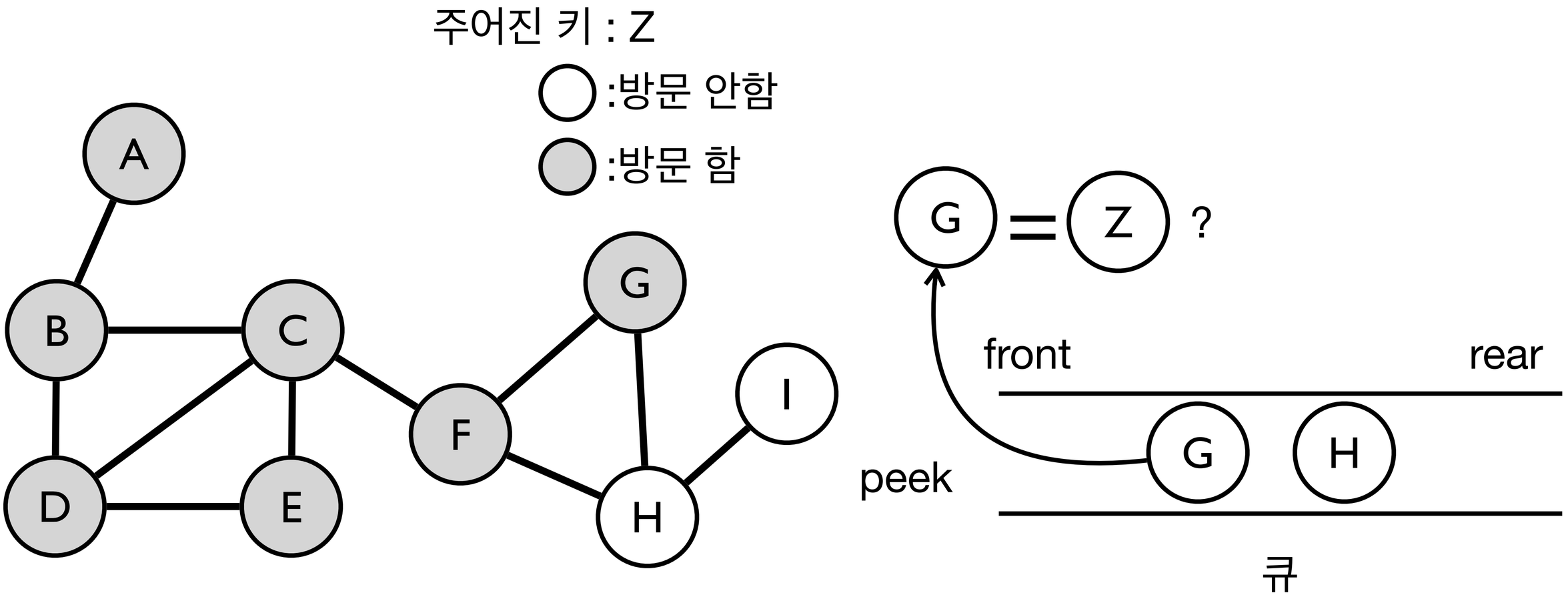


(1) 시작 노드를 큐에 enqueue 함

(2) 큐의 front의 노드가 찾고자 하는 노드인지 확인함 (찾았을 경우 해당 노드를 반환 후 종료).

(3) dequeue 후 dequeue한 노드의 인접 노드들 중 방문하지 않았으면서 큐에 없는 노드들을 큐에 삽입(enqueue)함

(4) 큐가 빌 때까지 (2), (3)을 반복함

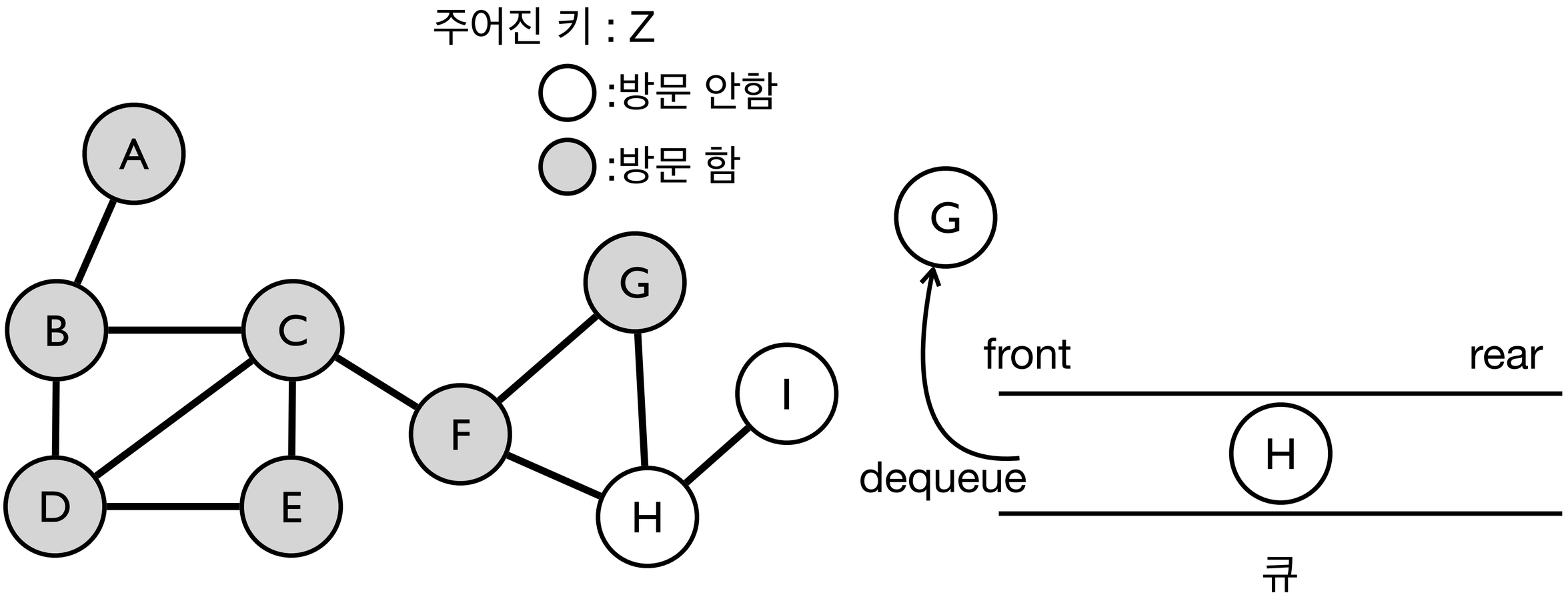


(1) 시작 노드를 큐에 enqueue 함

(2) 큐의 front의 노드가 찾고자 하는 노드인지 확인함 (찾았을 경우 해당 노드를 반환 후 종료).

(3) dequeue 후 dequeue한 노드의 인접 노드들 중 방문하지 않았으면서 큐에 없는 노드들을 큐에 삽입(enqueue)함

(4) 큐가 빌 때까지 (2), (3)을 반복함

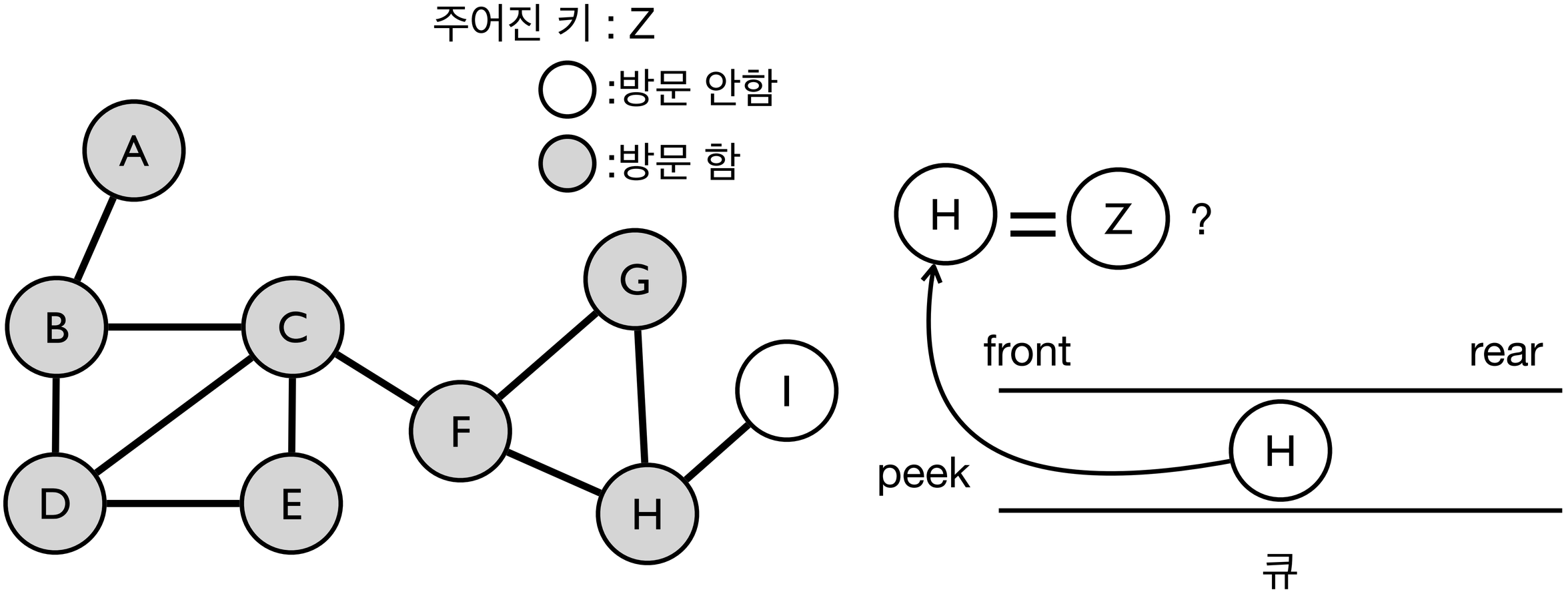


(1) 시작 노드를 큐에 enqueue 함

(2) 큐의 front의 노드가 찾고자 하는 노드인지 확인함 (찾았을 경우 해당 노드를 반환 후 종료).

(3) dequeue 후 dequeue한 노드의 인접 노드들 중 방문하지 않았으면서 큐에 없는 노드들을 큐에 삽입(enqueue)함

(4) 큐가 빌 때까지 (2), (3)을 반복함

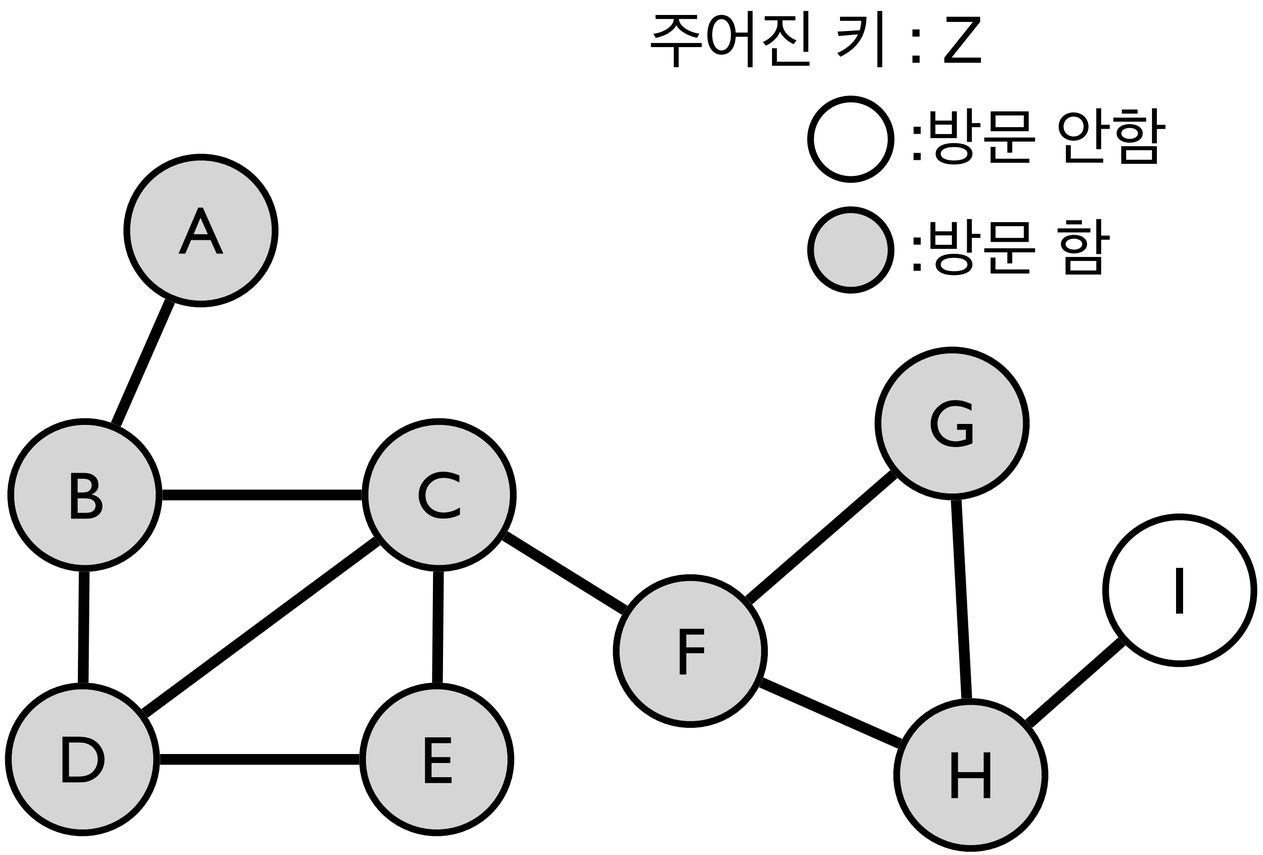


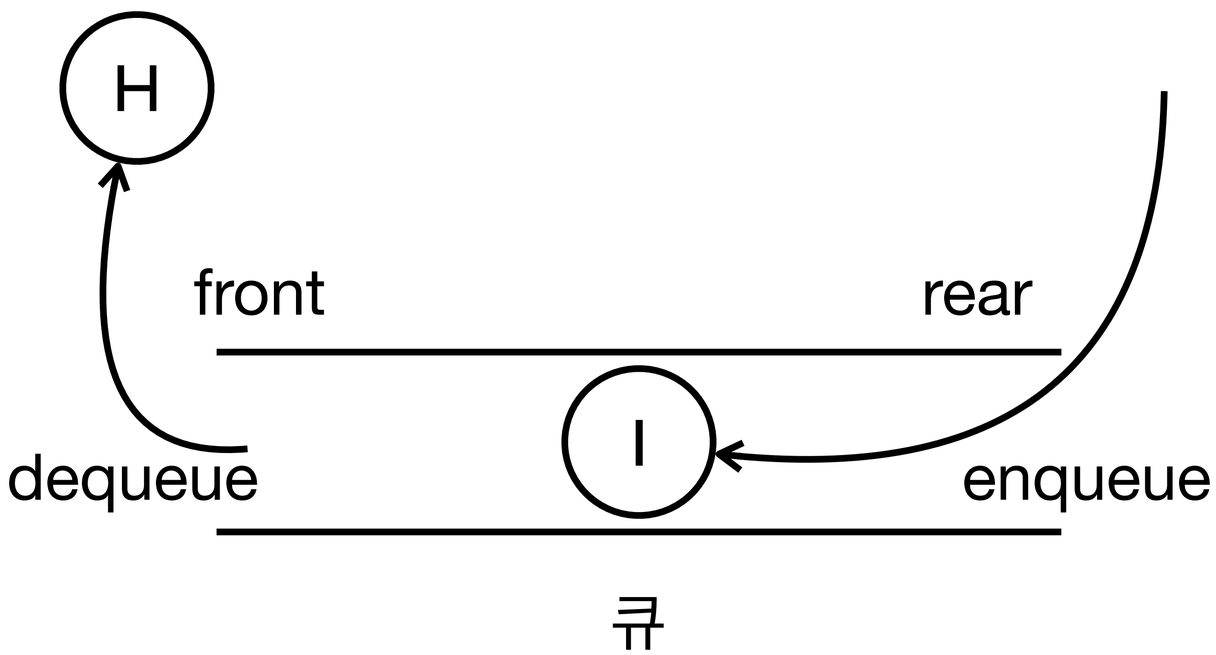
(1) 시작 노드를 큐에 enqueue 함

(2) 큐의 front의 노드가 찾고자 하는 노드인지 확인함 (찾았을 경우 해당 노드를 반환 후 종료).

(3) dequeue 후 dequeue한 노드의 인접 노드들 중 방문하지 않았으면서 큐에 없는 노드들을 큐에 삽입(enqueue)함

(4) 큐가 빌 때까지 (2), (3)을 반복함



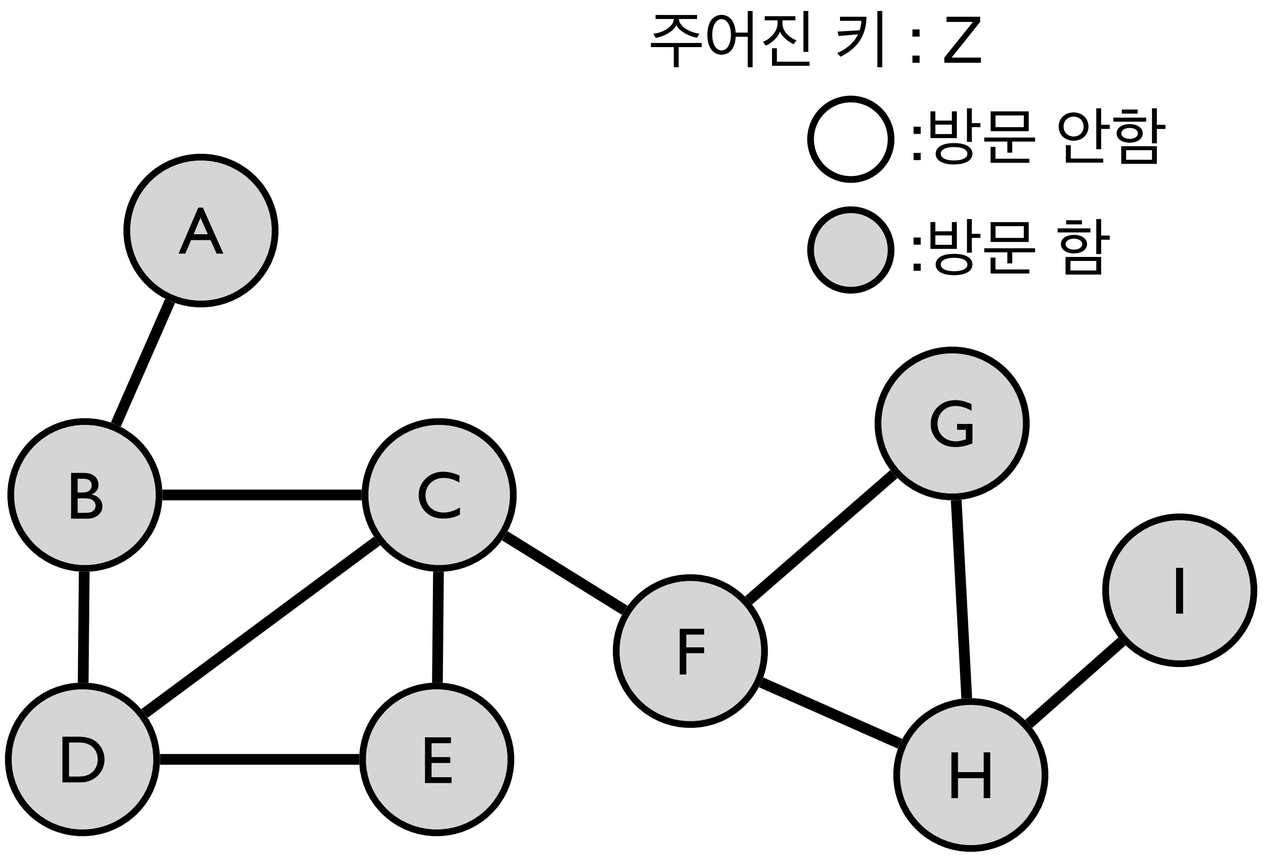


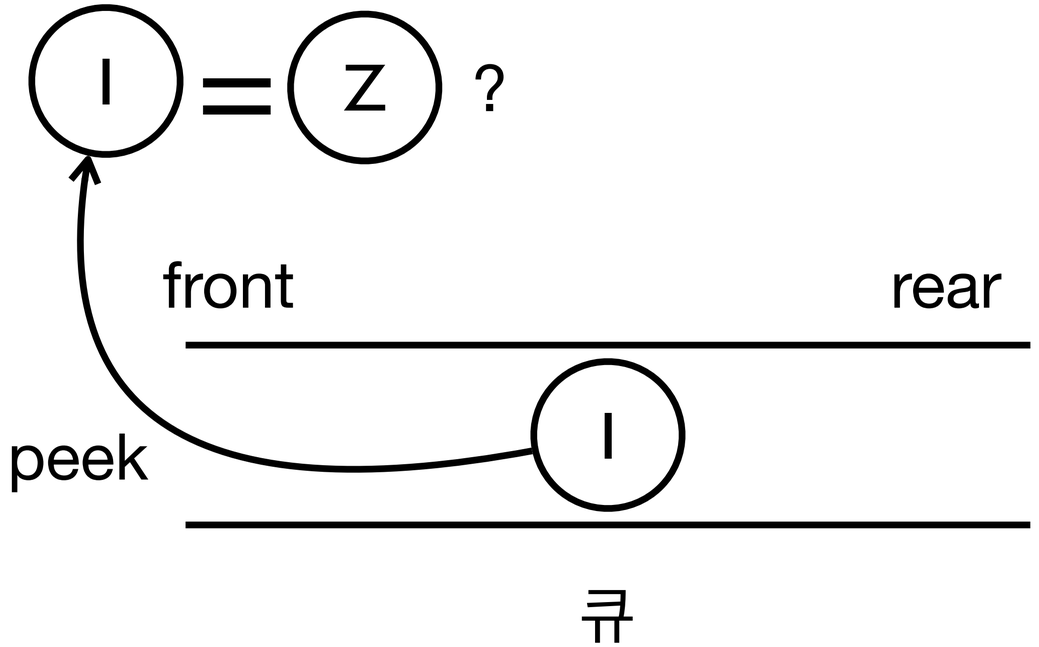
(1) 시작 노드를 큐에 enqueue 함

(2) 큐의 front의 노드가 찾고자 하는 노드인지 확인함 (찾았을 경우 해당 노드를 반환 후 종료).

(3) dequeue 후 dequeue한 노드의 인접 노드들 중 방문하지 않았으면서 큐에 없는 노드들을 큐에 삽입(enqueue)함

(4) 큐가 빌 때까지 (2), (3)을 반복함



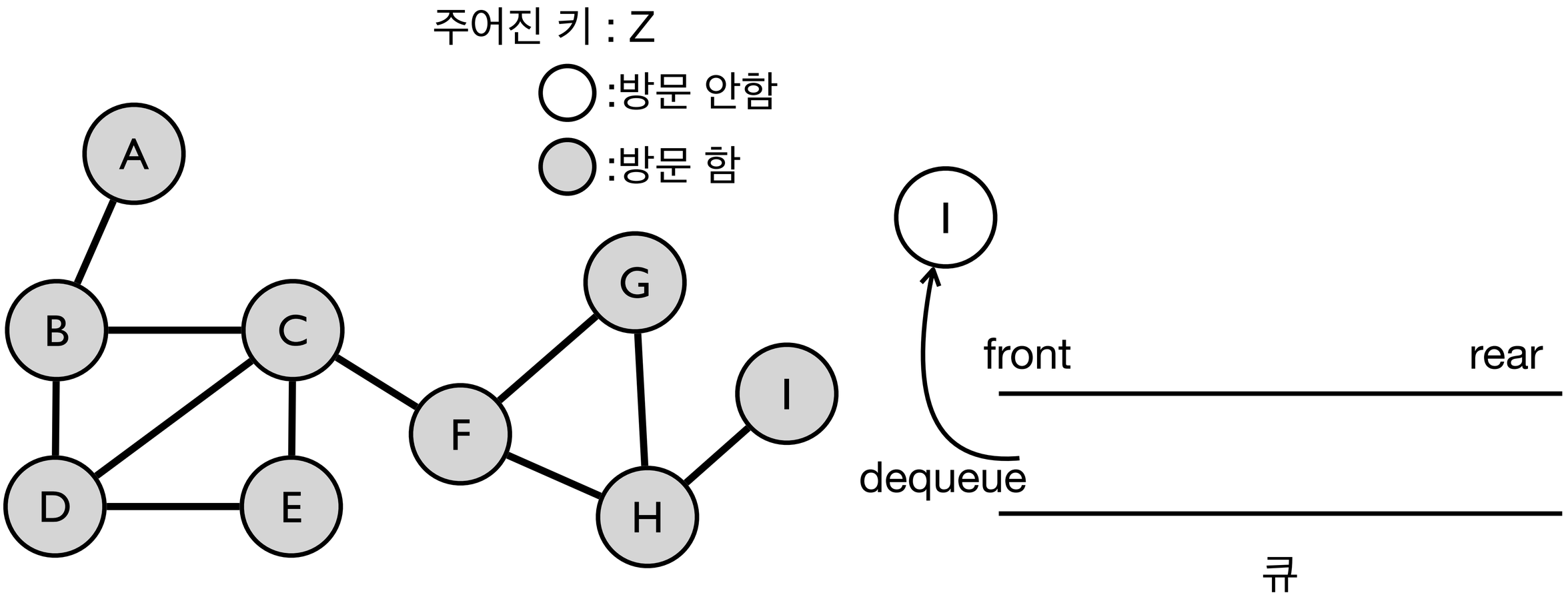


(1) 시작 노드를 큐에 enqueue 함

(2) 큐의 front의 노드가 찾고자 하는 노드인지 확인함 (찾았을 경우 해당 노드를 반환 후 종료).

(3) dequeue 후 dequeue한 노드의 인접 노드들 중 방문하지 않았으면서 큐에 없는 노드들을 큐에 삽입(enqueue)함

(4) 큐가 빌 때까지 (2), (3)을 반복함

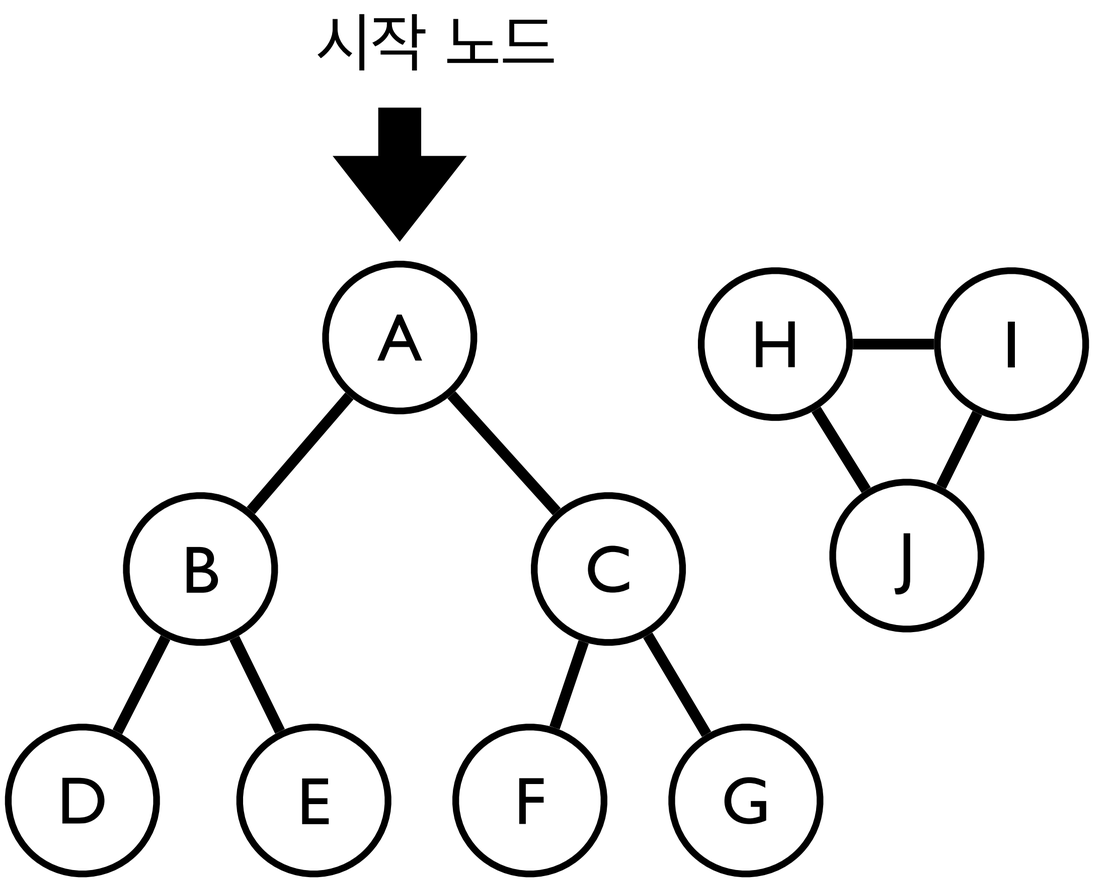


Example Code

```
#include "Queue.h"
int BFS(Graph *graph, int startNode, int targetValue) {
    if (startNode >= MAX_NODES || !graph->nodePresent[startNode]) {
        return -1;
    }
    bool visited[MAX_NODES] = { false };
    Queue* queue = create();
    enqueue(queue, startNode);
    while (!isEmpty(queue)) {
        int current = dequeue(queue);
        if (!visited[current]) {
            visited[current] = true;
            if (graph->data[current] == targetValue) {
                return current;
            }
            for (int i = 0; i < MAX_NODES; i++) {
                if (graph->adjacencyMatrix[current][i] && graph->nodePresent[i] && !visited[i]) {
                    enqueue(queue, i);
                }
            }
        }
    }
    return -1; // No such node
}
```

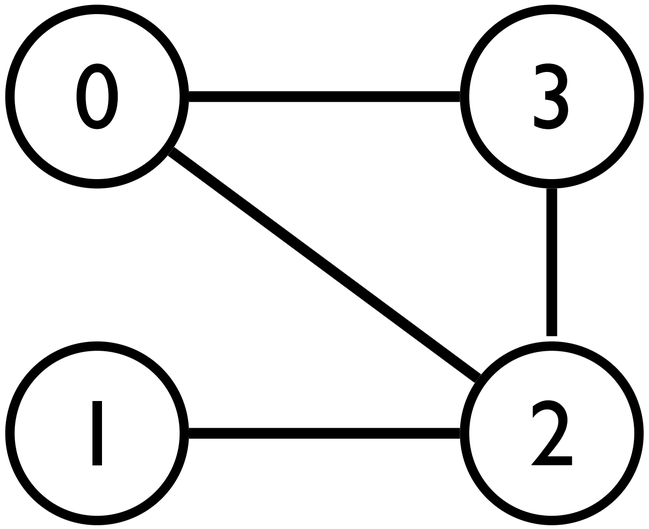
Example

- 왼쪽의 노드가 10개인 그래프 데이터에서 BFS순회를 하였을 때 방문 노드의 순서를 기술하시오



마무리

- 그래프는 데이터간의 (임의의) 관계를 표현할수 있는 자료구조
- 그래프 자료구조는 노드(vertex)와 간선(edge)들의 집합임



그래프 G = (V, E)

노드 (V) = {0, 1, 2, 3}

간선 (E) = {(0, 1), (0, 2), (0, 3), (1, 2), (2, 3)}