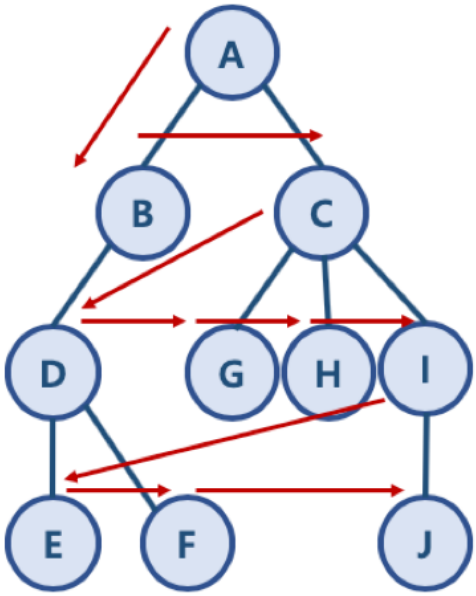
너비 우선 탐색 (Breadth-First Search)

1. BFS 와 DFS란 ?

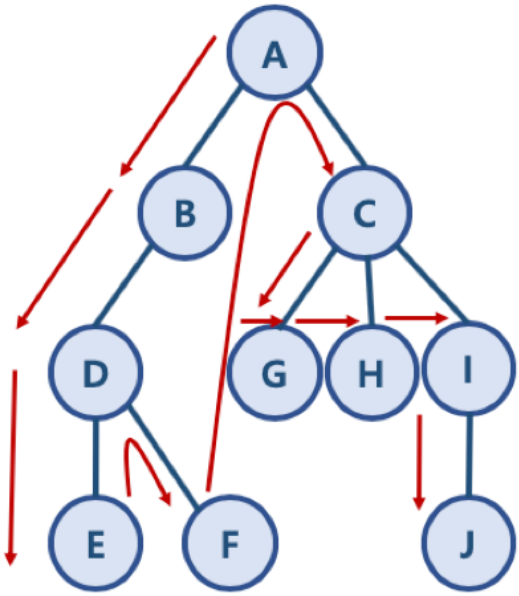
- 대표적인 그래프 **탐색** 알고리즘
  - 너비 우선 탐색(Breadth First Search) : 정점들과 같은 레벨에 있는 노드들 (형제 노드들)을 먼저 탐색하는 방식
  - 깊이 우선 탐색(Depth First Search) : 정점의 자식들을 먼저 탐색하는 방식

BFS/DFS 방식 이해를 위한 예제

- BFS 방식 : A - B - C - D - G - H - I - E - F - J
  - 한 단계씩 내려가면서, 해당 노드와 같은 레벨에 있는 노드들(형제 노드들)을 먼저 순회함
- DFS 방식 : A - B - D - E - F - C - G - H - I - J
  - 한 노드의 자식을 타고 끝까지 순회한 후, 다시 돌아와서 다른 형제들의 자식을 타고 내려가며 순회함



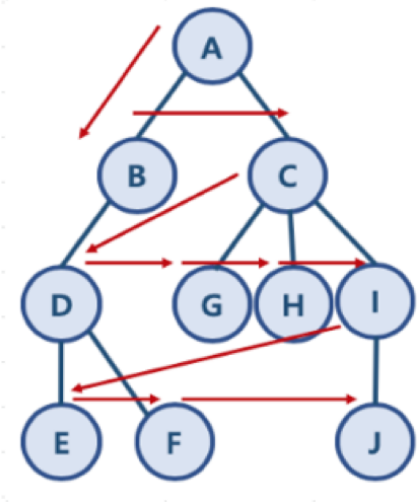
Breadth First Search



Depth First Search

2. 파이썬으로 그래프를 표현하는 방법

- 파이썬에서 제공하는 딕셔너리와 리스트 자료 구조를 활용해서 그래프를 표현할 수 있음



key	values				
A	B	C			
B	A	D			
C	A	G	H	I	
D	B	E	F		
E	D				
F	D				
G	C				
H	C				
I	C	J			
J	I				

```
graph = dict()

graph['A'] = ['B', 'C']
graph['B'] = ['A', 'D']
graph['C'] = ['A', 'G', 'H', 'I']
graph['D'] = ['B', 'E', 'F']
```

```
graph['E'] = ['D']
graph['F'] = ['D']
graph['G'] = ['C']
graph['H'] = ['C']
graph['I'] = ['C', 'J']
graph['J'] = ['I']
```

```
graph
>>
{'A': ['B', 'C'],
 'B': ['A', 'D'],
 'C': ['A', 'G', 'H', 'I'],
 'D': ['B', 'E', 'F'],
 'E': ['D'],
 'F': ['D'],
 'G': ['C'],
 'H': ['C'],
 'I': ['C', 'J'],
 'J': ['I']}
```

3. BFS 알고리즘 구현

- 자료구조 큐를 활용함
    - need_visit 큐와 visited 큐, 두 개의 큐를 생성

	visited queue (The visited list)									
	A	B	C	D	G	H	I	E	F	J
	need_visit queue (The list to need to visit)									
	A									
A	B	C								
B	C	A	D							
C	A	D	A	G	H	I				
D	A	G	H	I	B	E	F			
G	H	I	B	E	F	C				
H	I	B	E	F	C	C				
I	B	E	F	C	C	C	J			
E	F	C	C	C	J	D				
F	C	C	C	J	D	D				
J	D	D	I							

```
def bfs(graph, start_node):
  visited = list()
```

```
  need_visit = list()

  need_visit.append(start_node)

  while need_visit:
    node = need_visit.pop(0)
    if node not in visited:
      visited.append(node)
      need_visit.extend(graph[node])
  return visited
```

```
bfs(graph, 'A')
>>['A', 'B', 'C', 'D', 'G', 'H', 'I', 'E', 'F', 'J']
```

4. 시간 복잡도

- 일반적인 BFS 시간 복잡도
  - 노드 수 : V
  - 간선 수 : E
    - 위 코드에서 while need_visit은 V + E번 만큼 수행함
  - 시간 복잡도 : O(V+E)

```
def bfs(graph, start_node):
    visited = list()
    need_visit = list()

    need_visit.append(start_node)
    count = 0
    while need_visit:
        count += 1
        node = need_visit.pop(0)
        if node not in visited:
            visited.append(node)
            need_visit.extend(graph[node])
    print (count)
    return visited
```

```
bfs(graph, 'A')
>>19
['A', 'B', 'C', 'D', 'G', 'H', 'I', 'E', 'F', 'J']
```