SWIFT
Data Structure - Graph(BFS)

Bill Kim(김정훈) | ibillkim@gmail.com

목차

BFS

Concept

Examples

Implementation

References

BFS

BFS(Bredth-First Search)란 그래프(Graph)에서 노드를 탐색하기 위한 하나의 알고리즘 방법으로서 너비 우선 검색이라는 방식입니다.

BFS를 사용하여 최단 및 최소 경로를 구할 수 있습니다.

BFS는 큐를 통하여 최소한의 비용이 필요한 경로를 구합니다.

모든 곳을 탐색하는 것보다 최소 비용이 우선일 경우 본 알고리즘이 적합합니다.

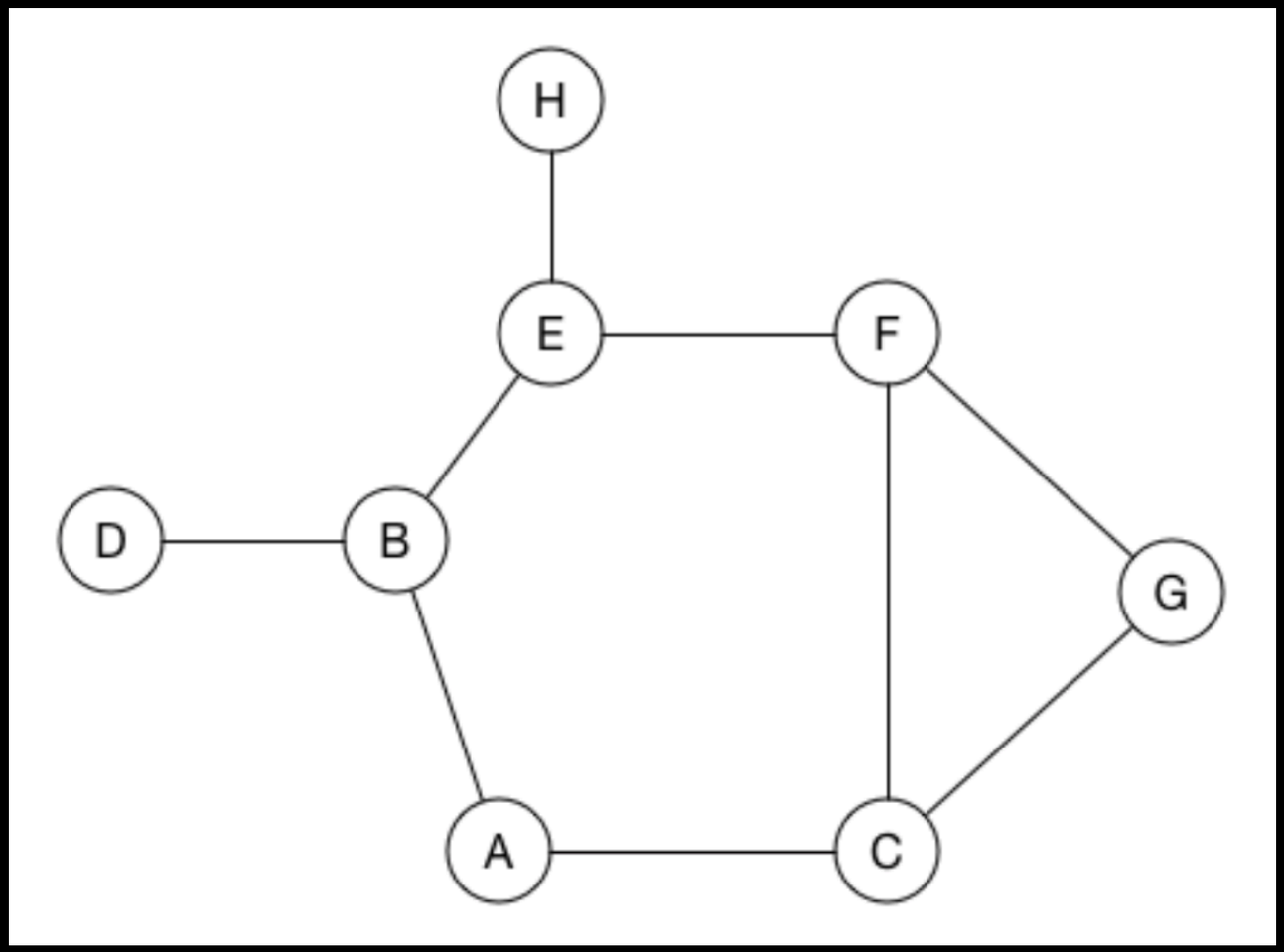
Concept

BFS의 기본적인 알고리즘 흐름은 다음과 같습니다.

1. 시작할 노드를 Queue에 넣는다.(Enqueue)
2. Queue에서 노드를 하나 꺼낸다.(Dequeue)
3. Dequeue한 노드와 인접한 노드가 있는지 확인한다.
4. Dequeue한 노드를 출력한다.
5. Dequeue한 노드와 인접한 노드를 Queue에 Enqueue한다.
   이미 방문한 노드는 Enqueue 하지 않는다.
6. 2번 부터 5번까지의 과정을 Queue가 비워질때까지 반복한다.

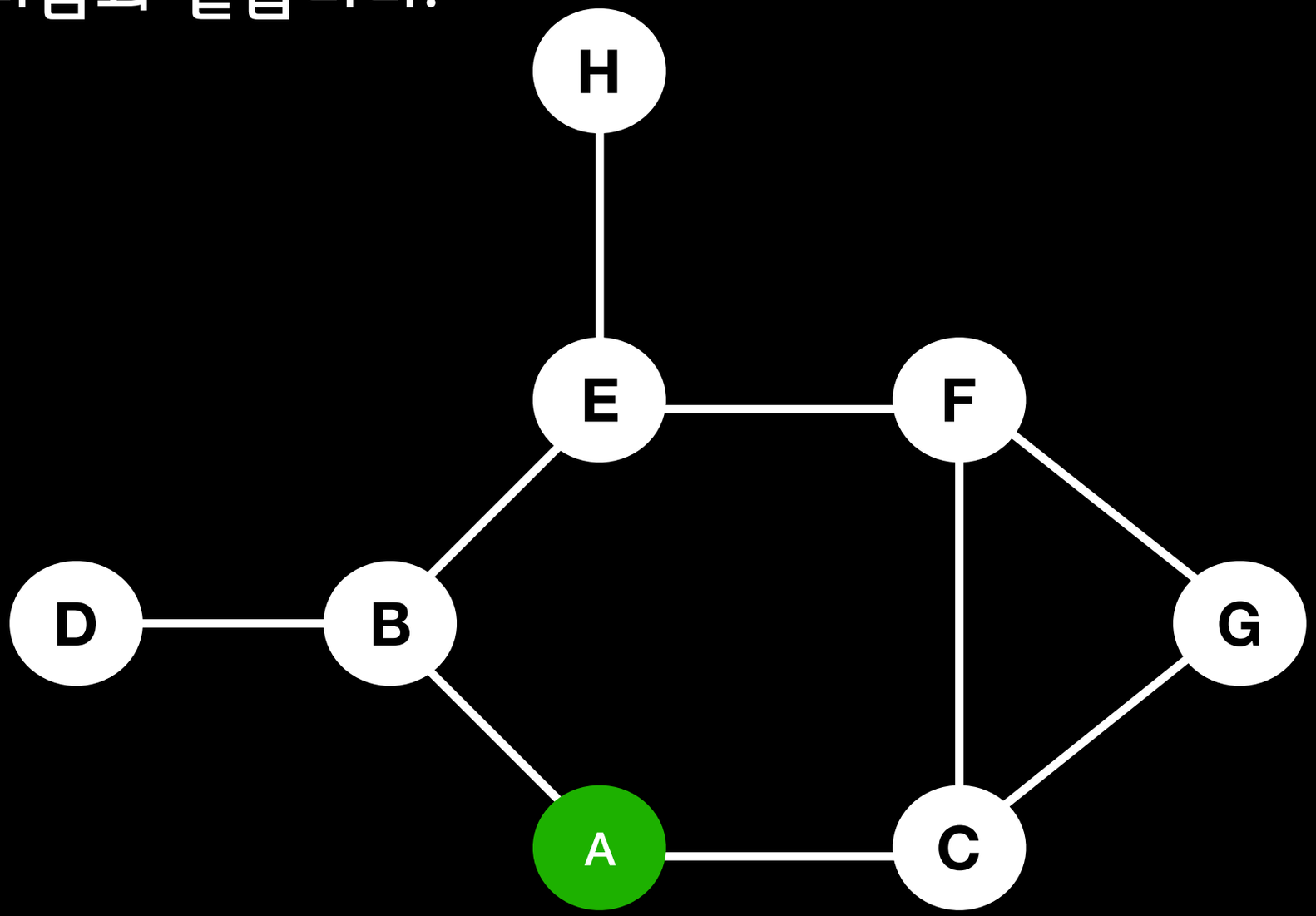
Examples

앞서 설명한 알고리즘을 실제 예제를 통해서 살펴봅니다.



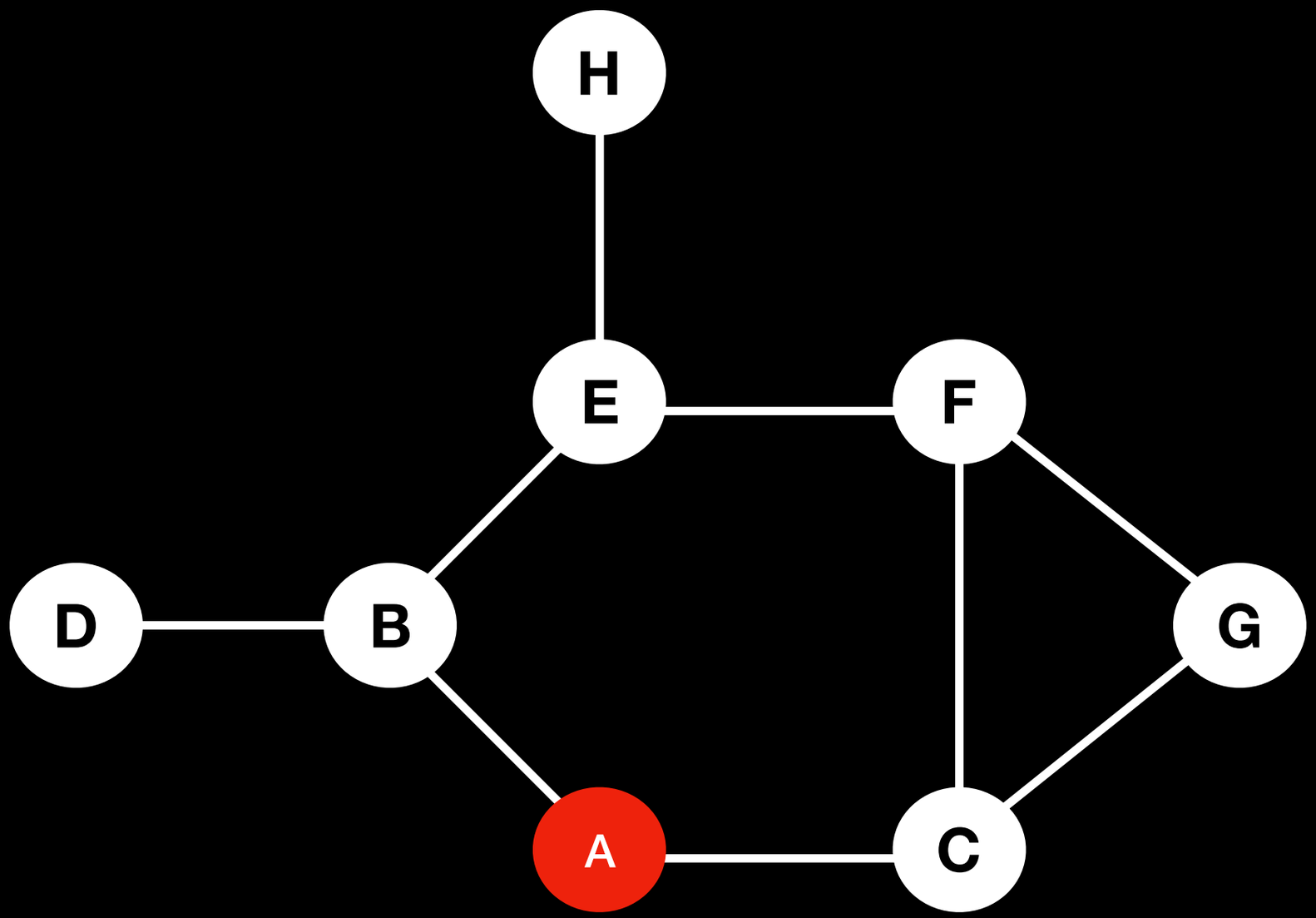
Examples

BFS 알고리즘을 통하여 A 노드에서 시작한 경로에 대한 과정을 살펴보면 다음과 같습니다.



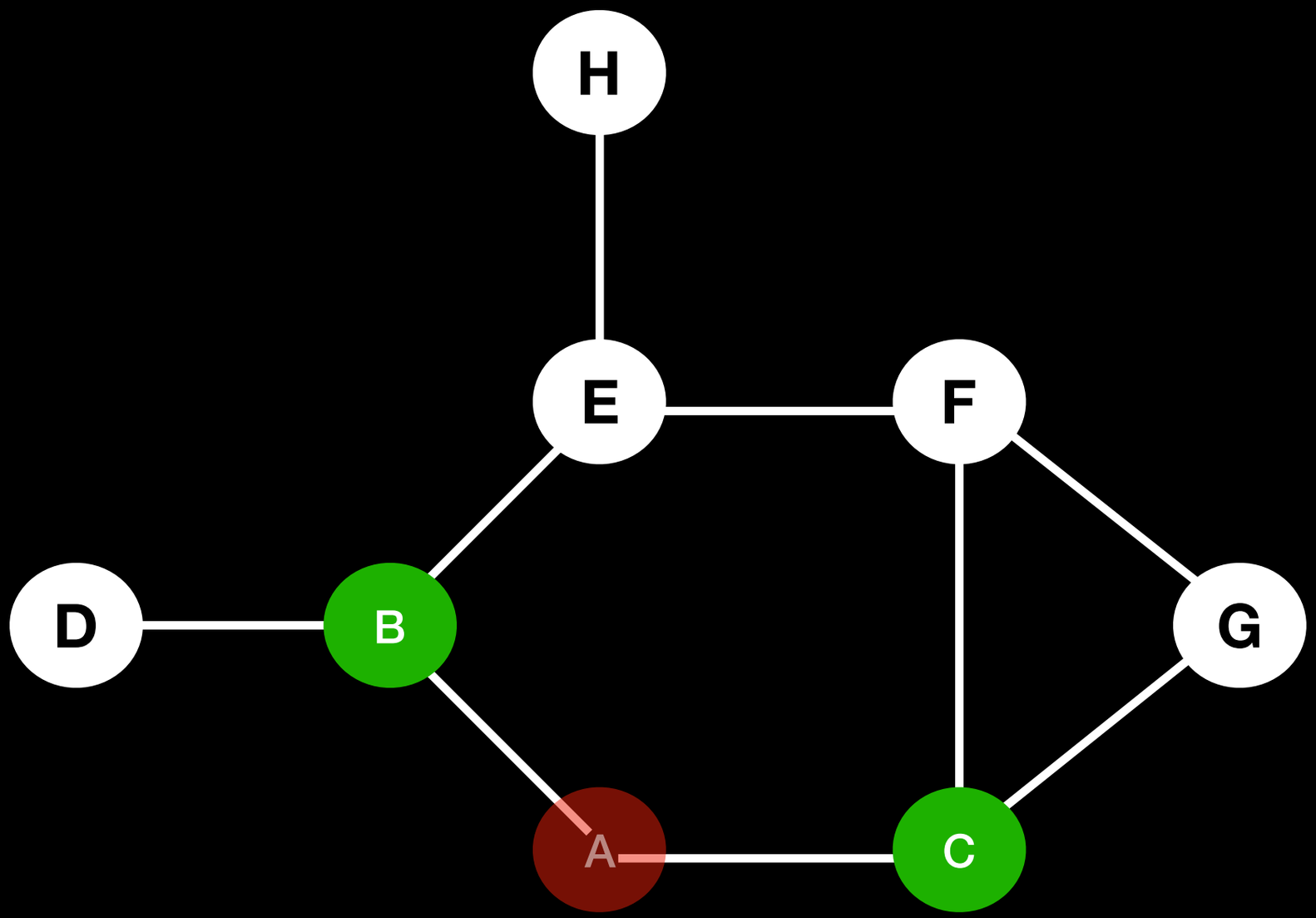
A 노드 Enqueue (Queue [A])

Examples



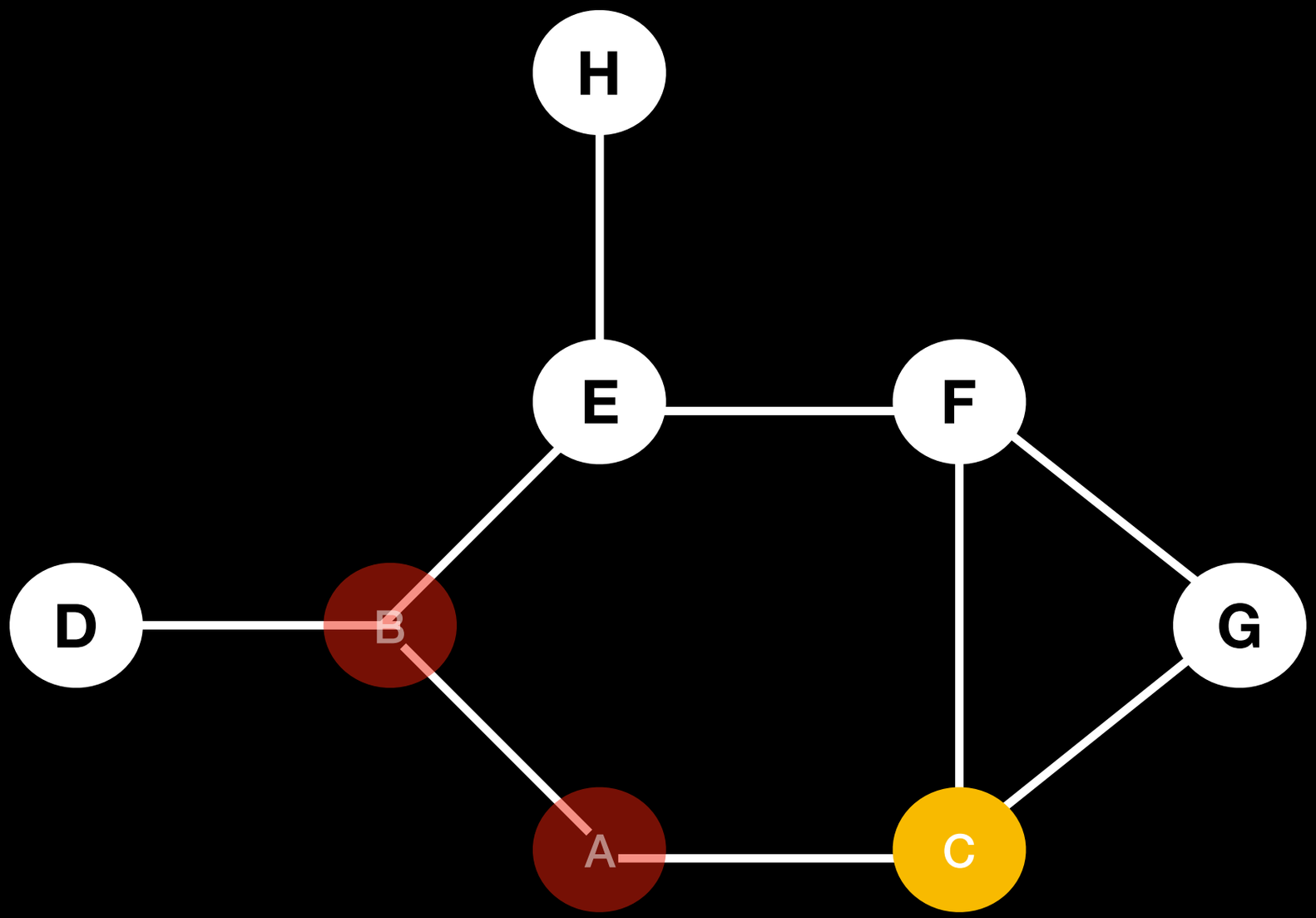
∧ 노드 Dequeue (Queue [ ]

Examples



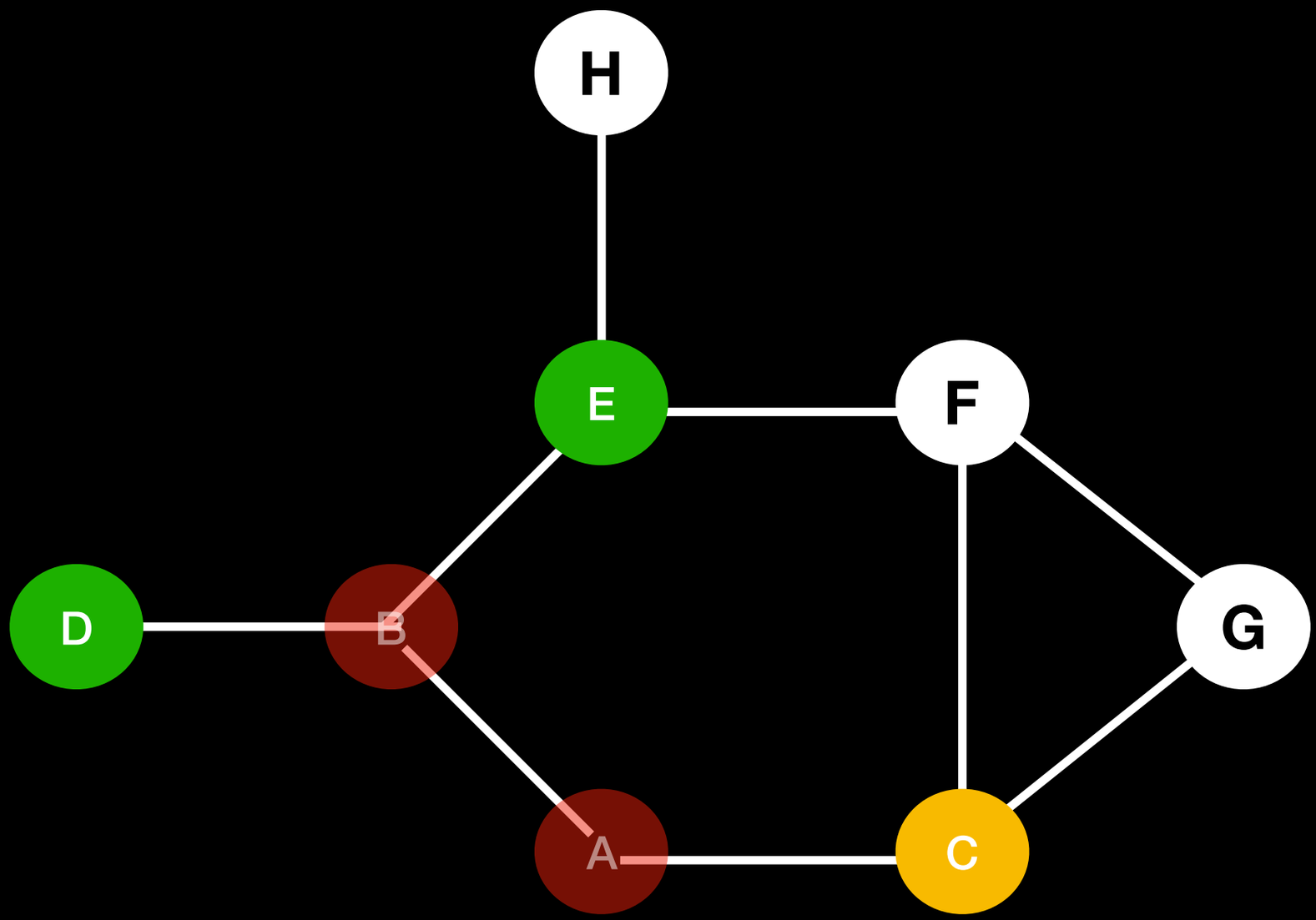
B, C 노드 Enqueue (Queue [B, C])

Examples



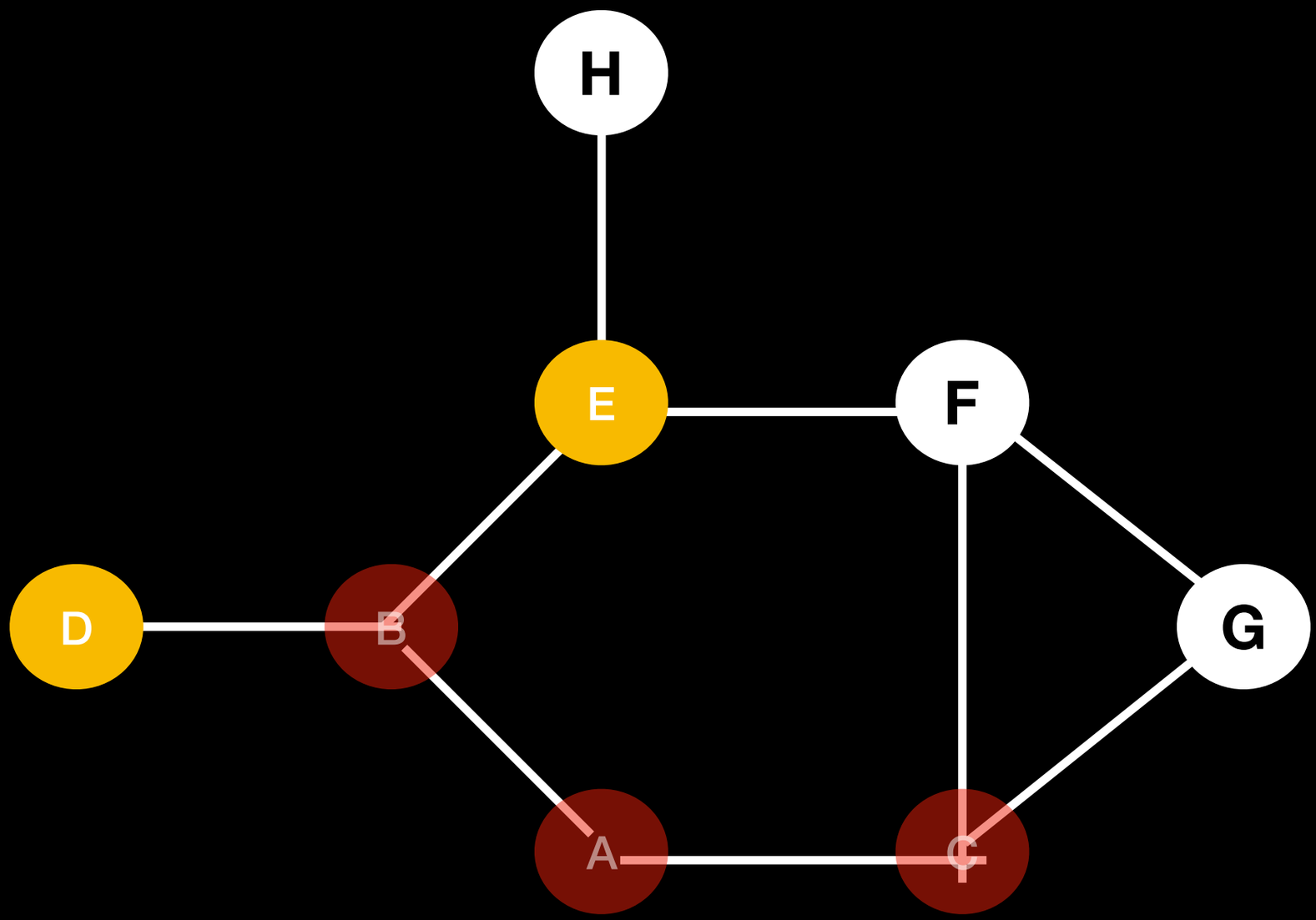
B 노드 Dequeue (Queue [C])

Examples



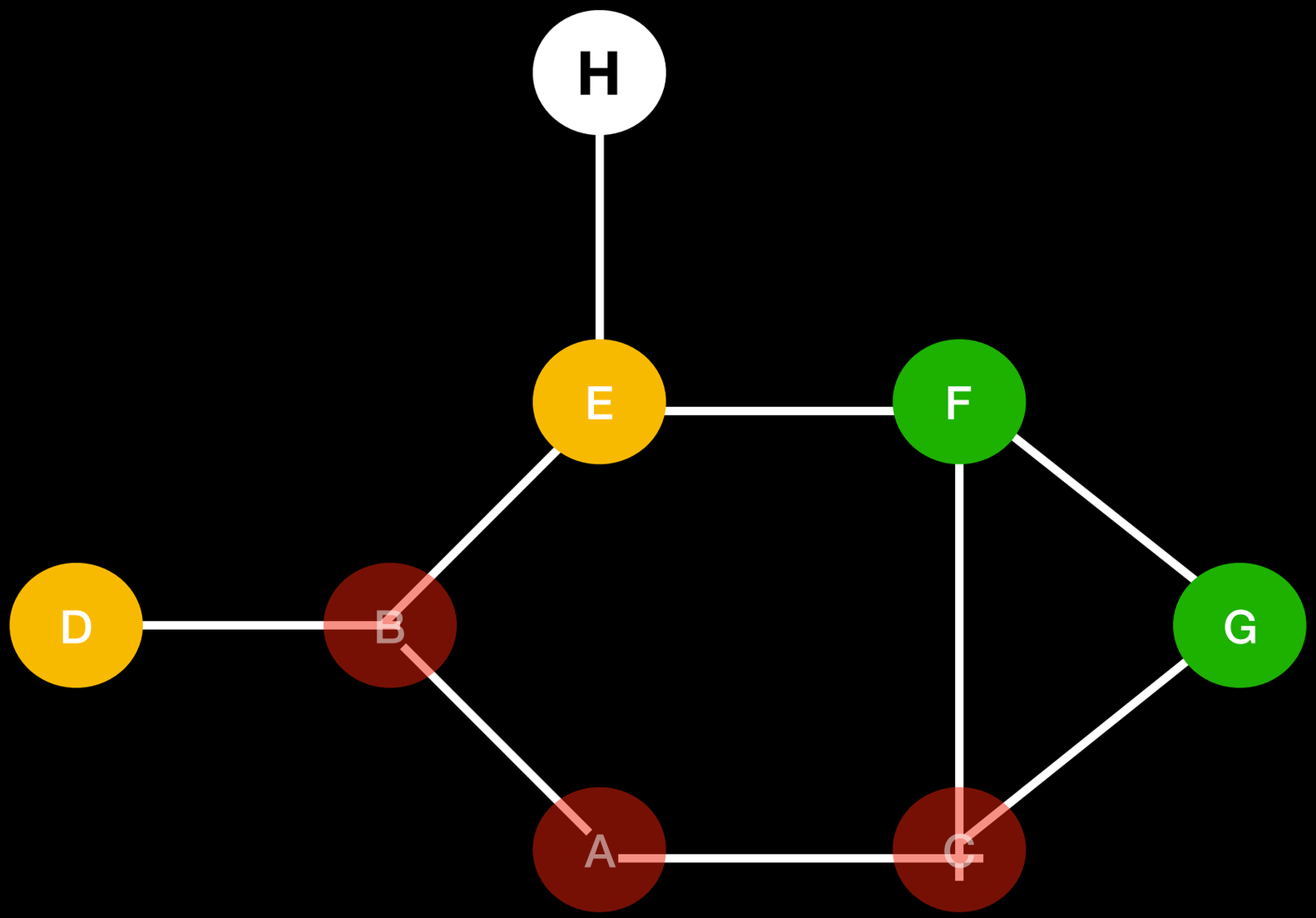
D, E 노드 Enqueue (Queue [C, D, E])

Examples



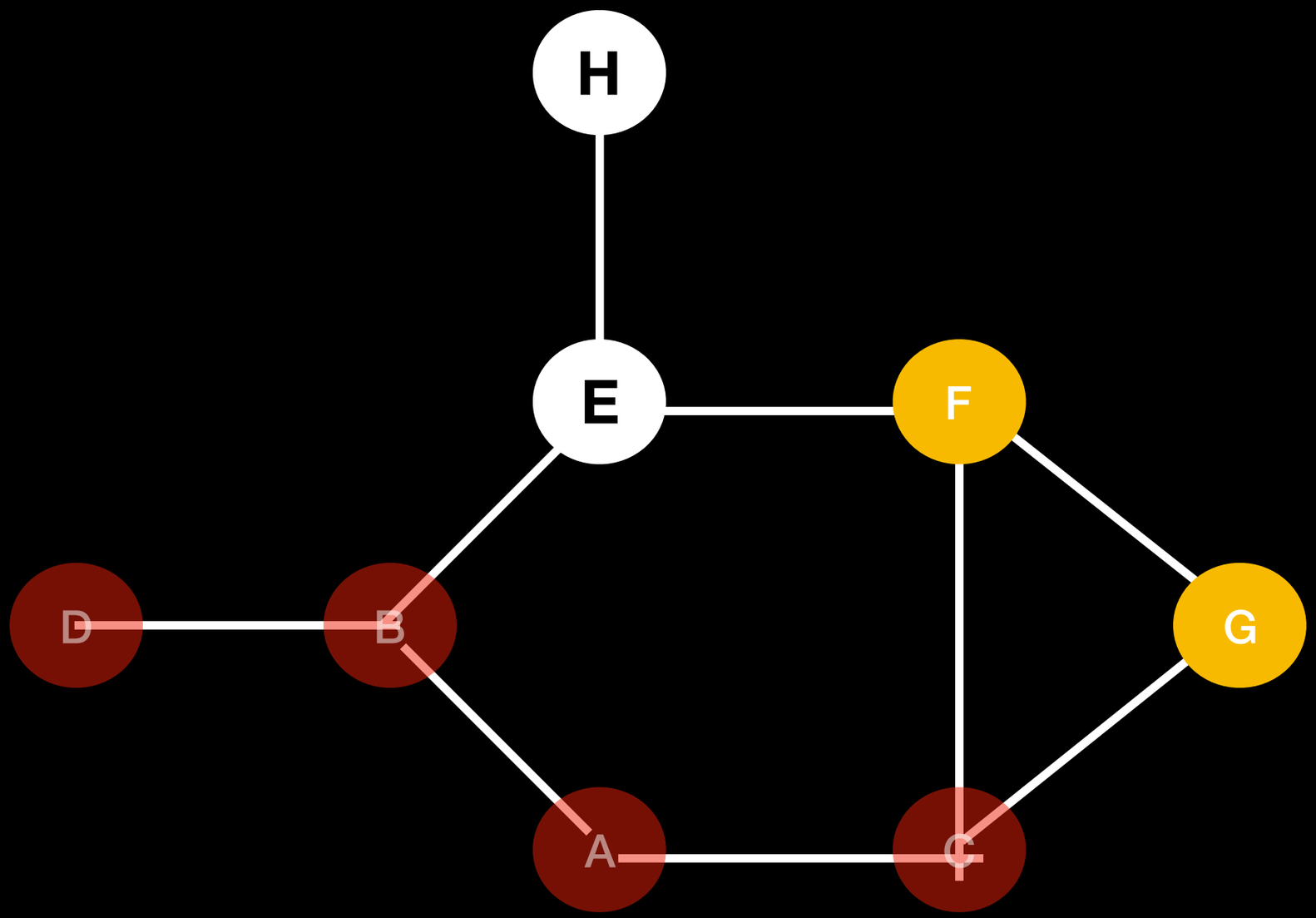
C 노드 Dequeue (Queue [D, E])

Examples



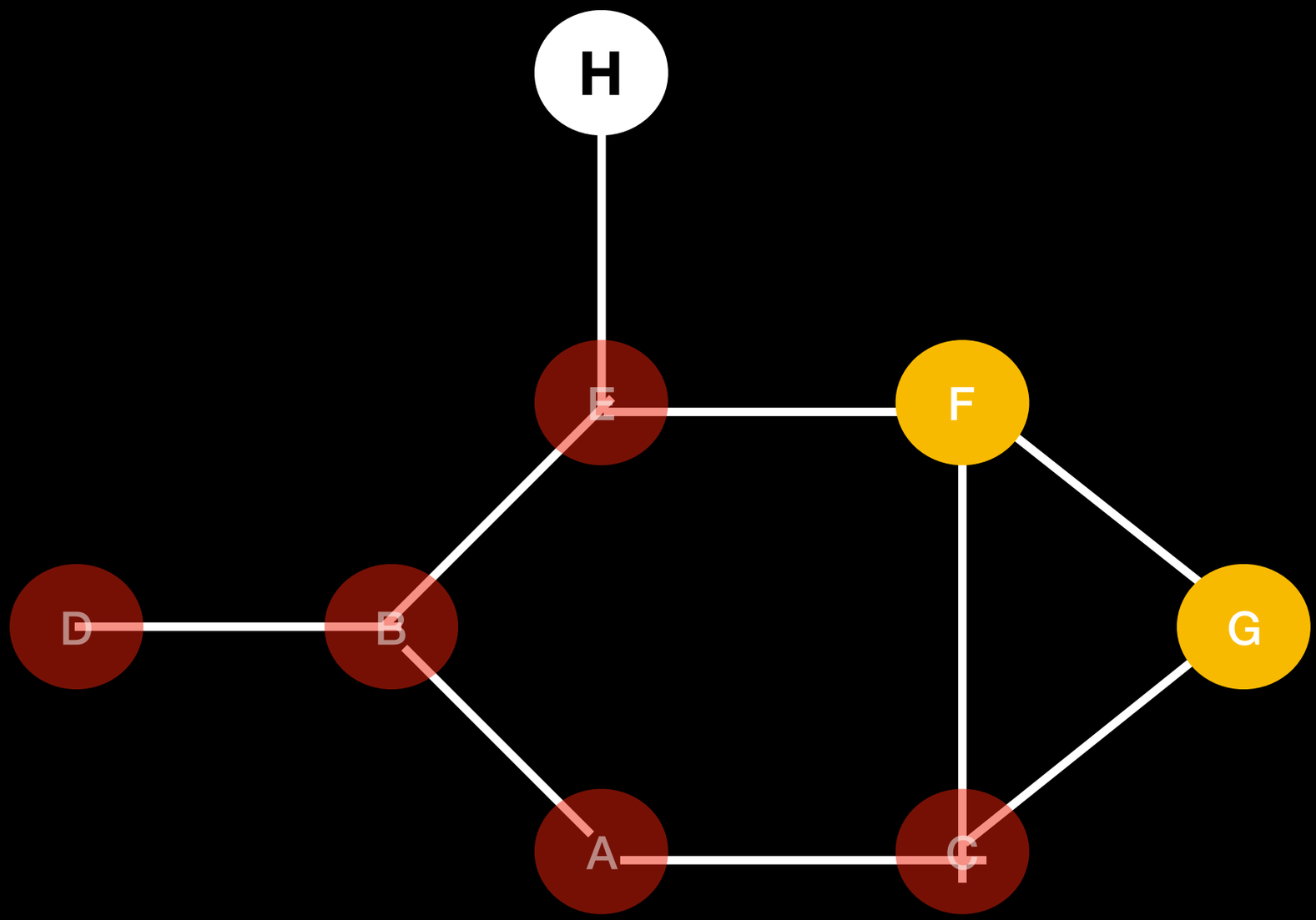
F, G 노드 Enqueue (Queue [D, E, F, G])

Examples



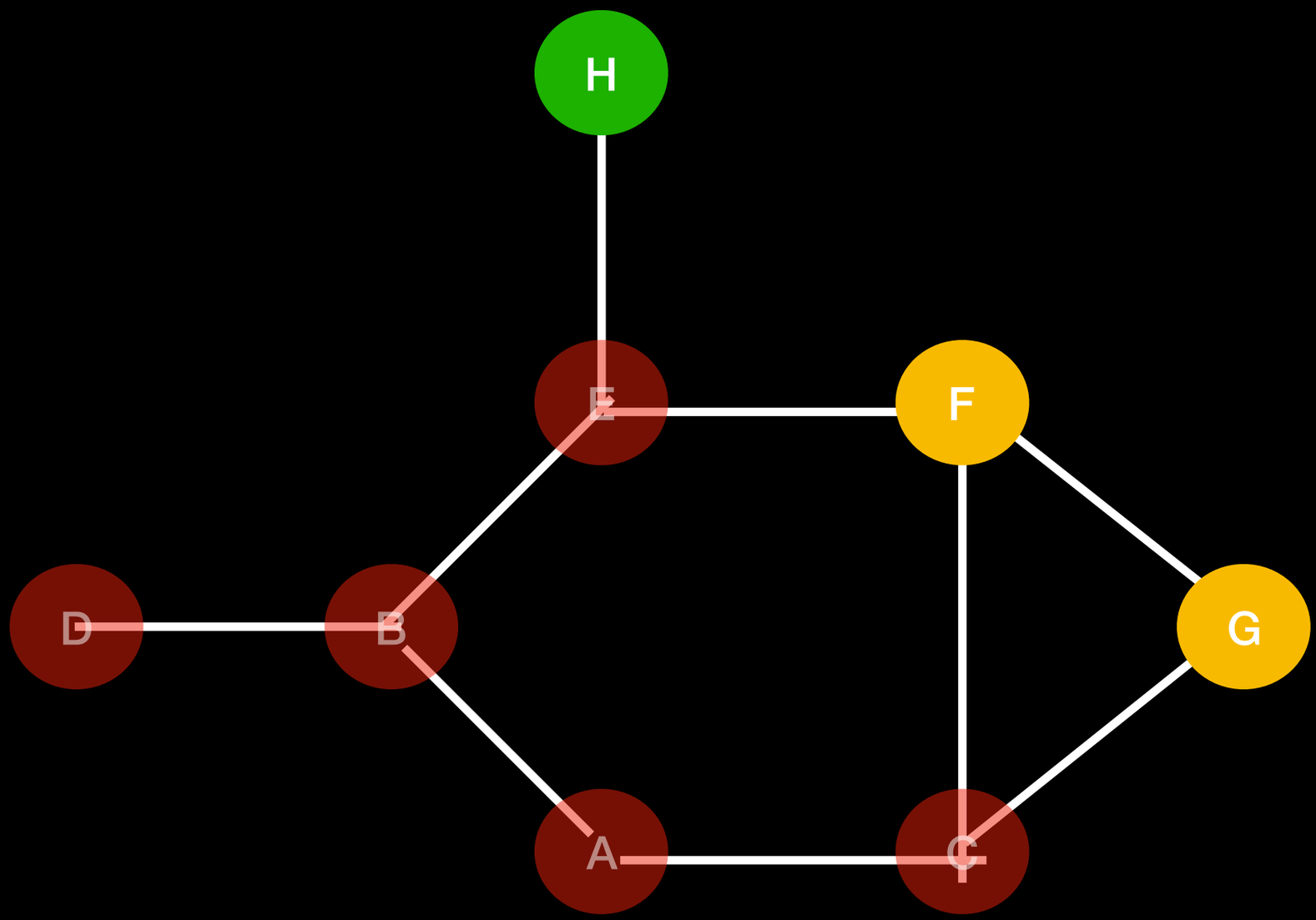
D 노드 Dequeue (Queue [E, F, G])

Examples



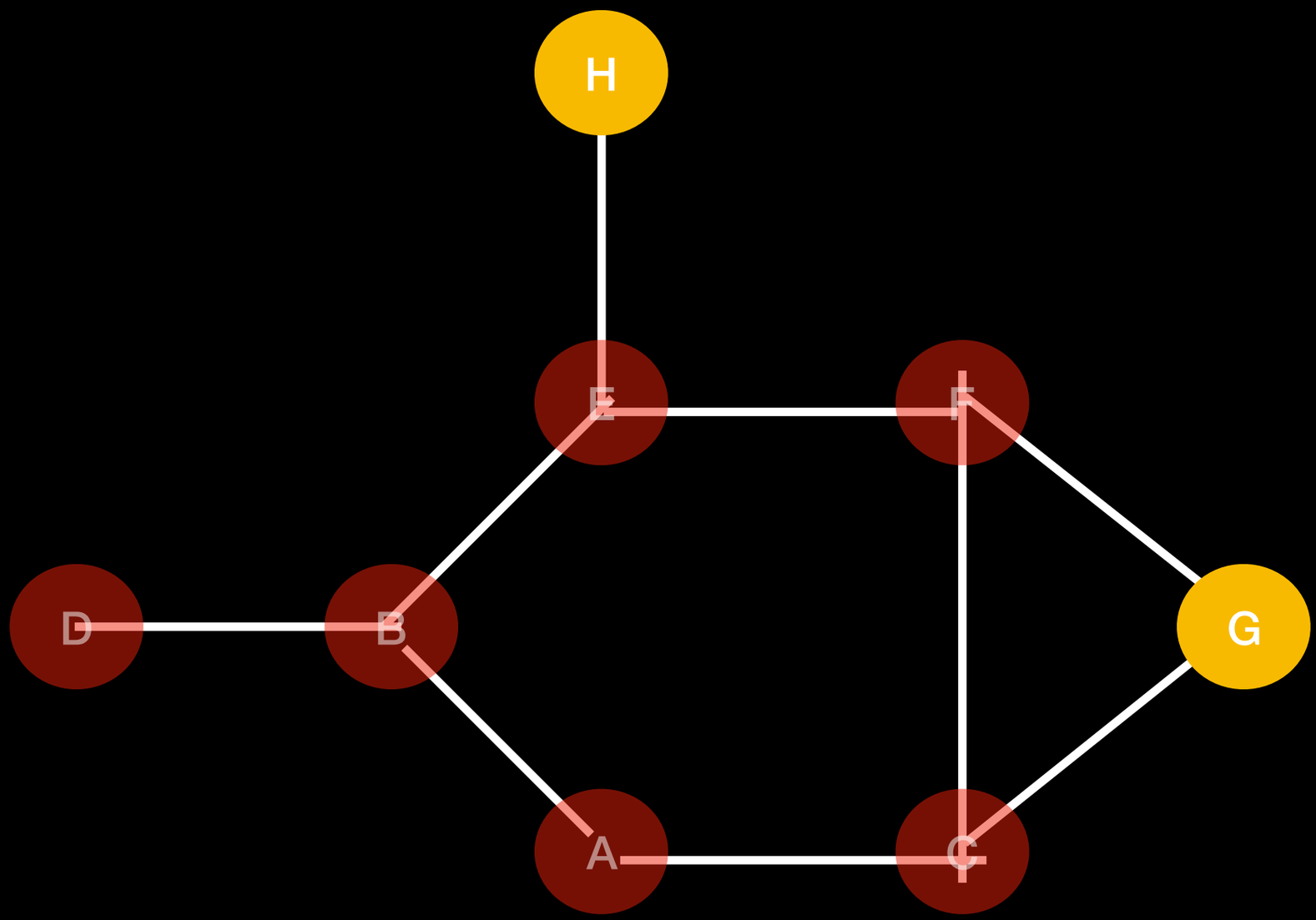
E 노드 Dequeue (Queue [F, G])

Examples



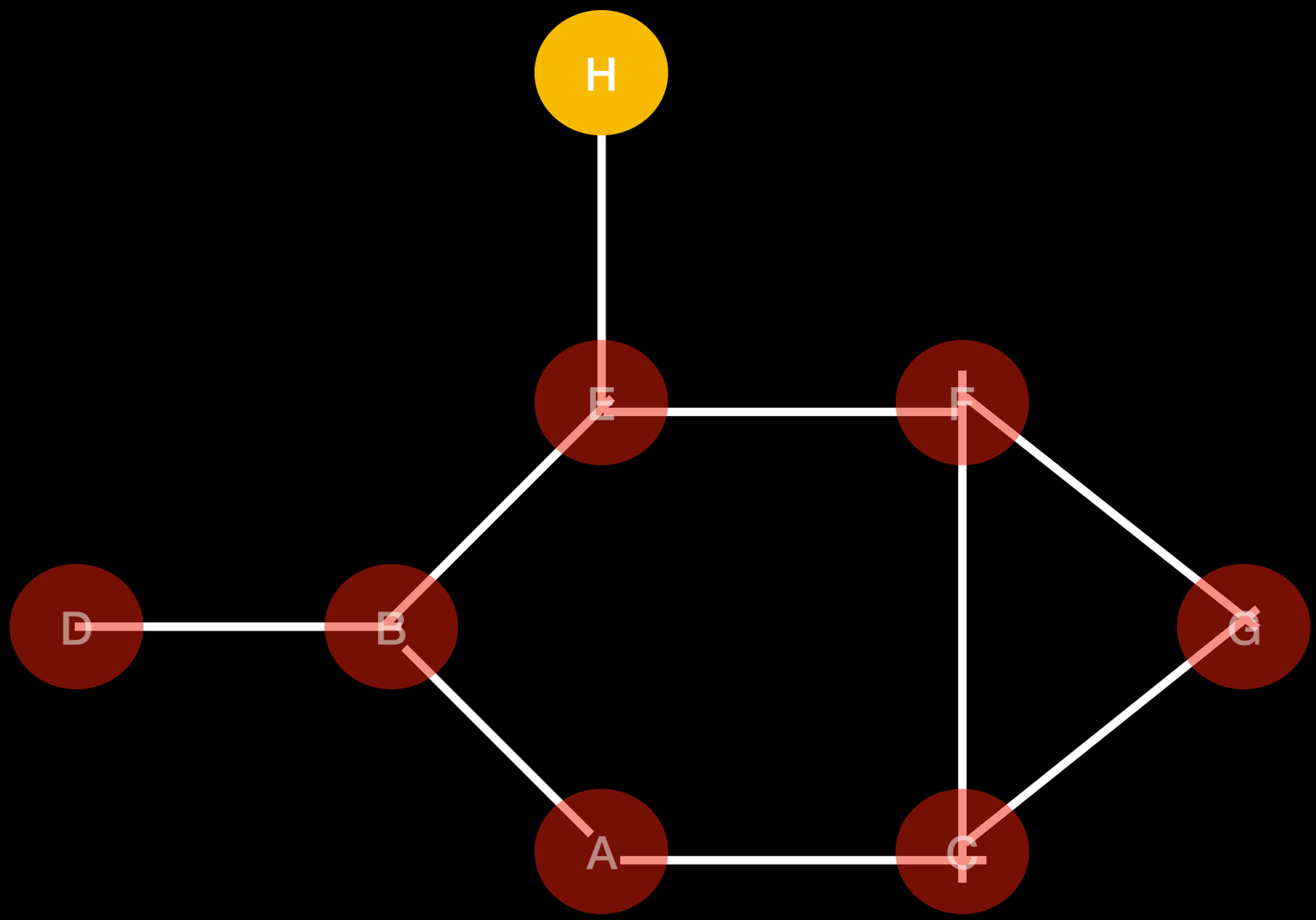
H 노드 Enqueue (Queue [F, G, H])

Examples



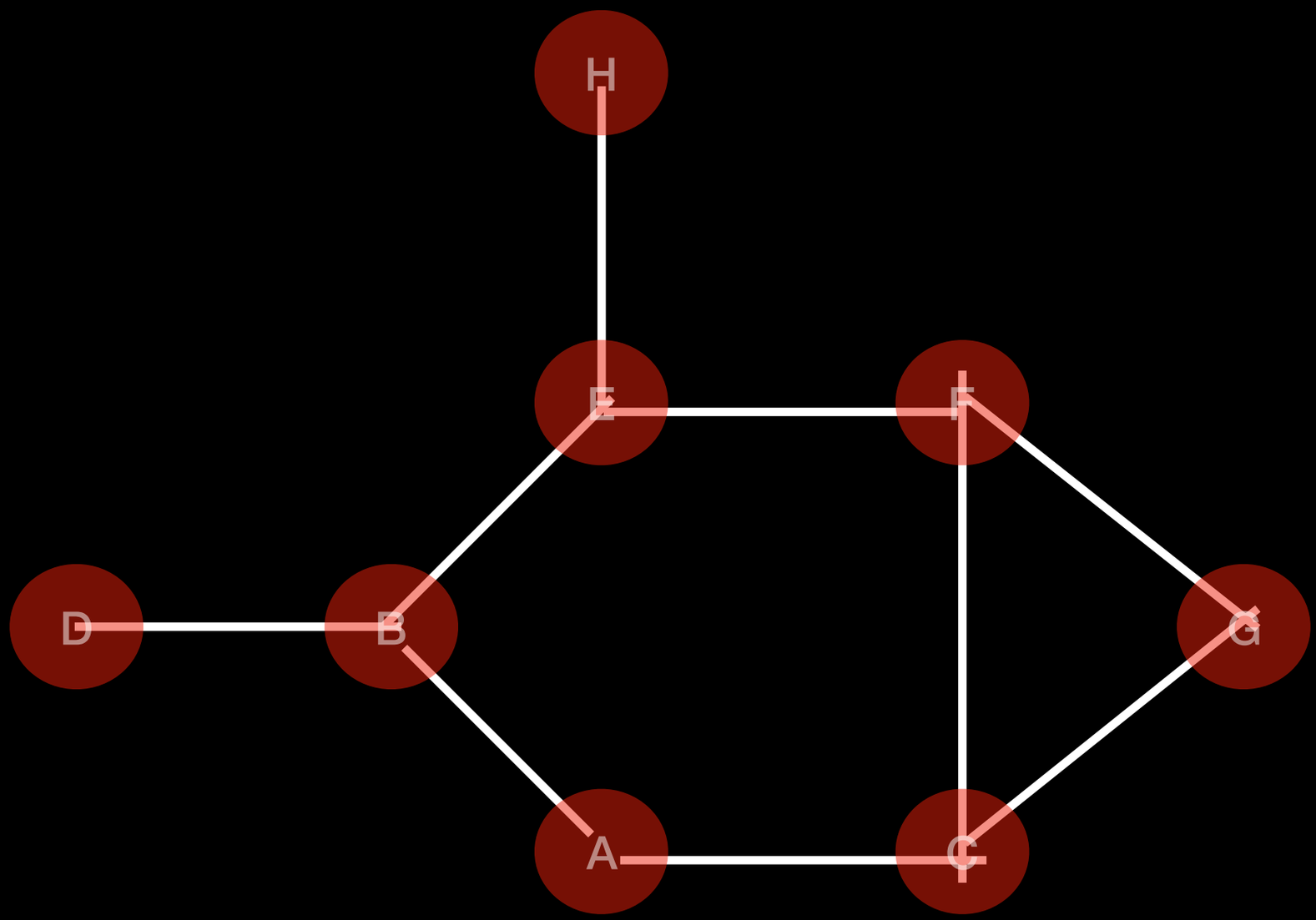
F 노드 Dequeue (Queue [G, H])

Examples



G 노드 Dequeue (Queue [H])

Examples



H 노드 Dequeue (Queue [ ])

Examples

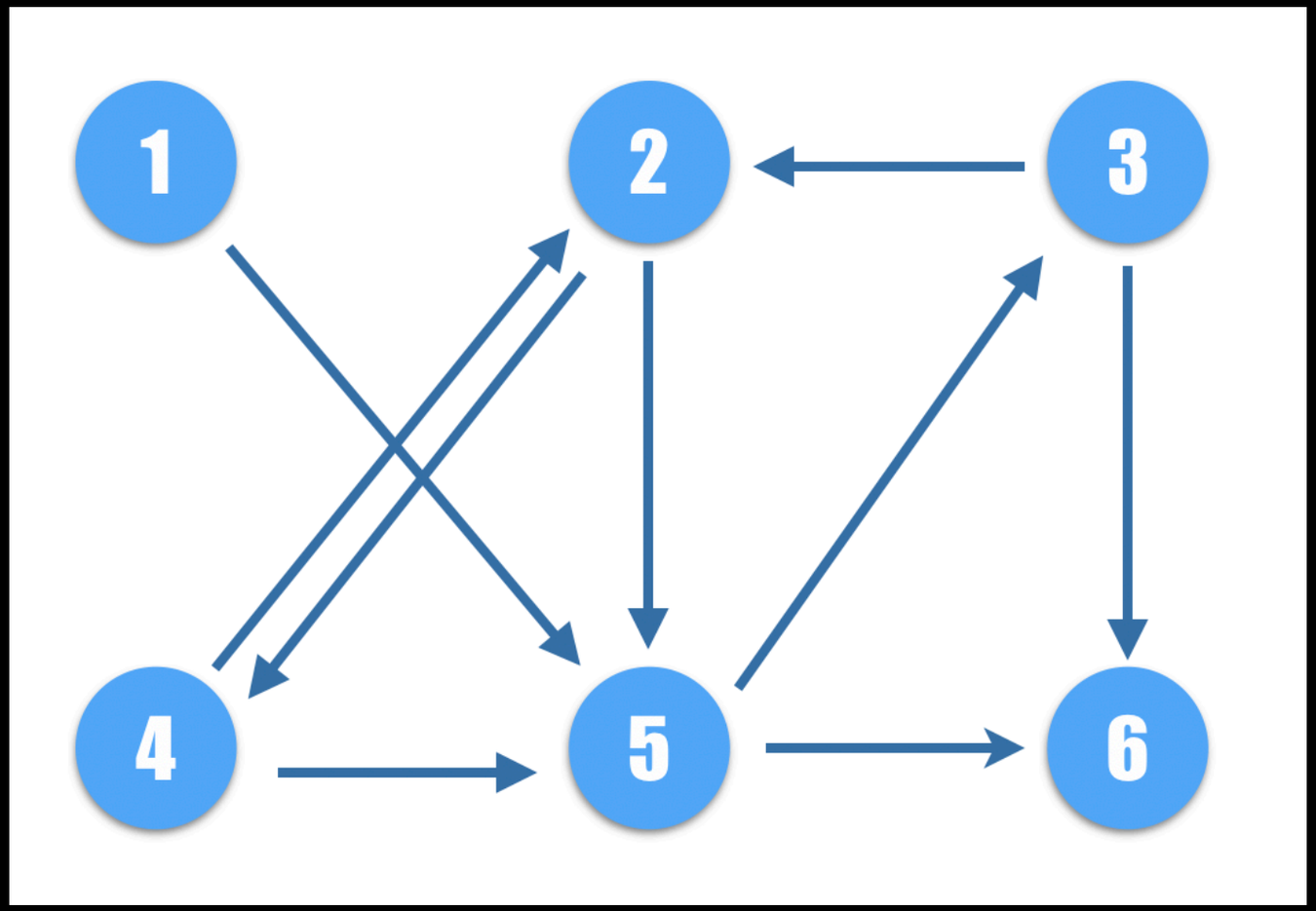
최종 경로를 살펴보면 아래와 같습니다.

```
// 시작할 노드인 a를 Queue에 넣습니다.
// Enqueue(+) : a - Queue [ a ]

//
// Dequeue(-) : a - Queue [ ] => a
// Enqueue(+) : b - Queue [ b ]
// Enqueue(+) : c - Queue [ b, c ]
// Dequeue(-) : b - Queue [ c ] => a, b
// Enqueue(+) : d - Queue [ c, d ]
// Enqueue(+) : e - Queue [ c, d, e ]
// Dequeue(-) : c - Queue [ d, e ] => a, b, c
// Enqueue(+) : f - Queue [ d, e, f ]
// Enqueue(+) : g - Queue [ d, e, f, g ]
// Dequeue(-) : d - Queue [ e, f, g ] => a, b, c, d
// Dequeue(-) : e - Queue [ f, g ] => a, b, c, d, e
// Enqueue(+) : h - Queue [ f, g, h ]
// Dequeue(-) : f - Queue [ g, h ] => a, b, c, d, e, f
// Dequeue(-) : g - Queue [ h ] => a, b, c, d, e, f, g
// Dequeue(-) : h - Queue [ ] => a, b, c, d, e, f, g, h
```

최종 경로 : [“a", "b", "c", "d", "e", "f", "g", "h"]

Examples

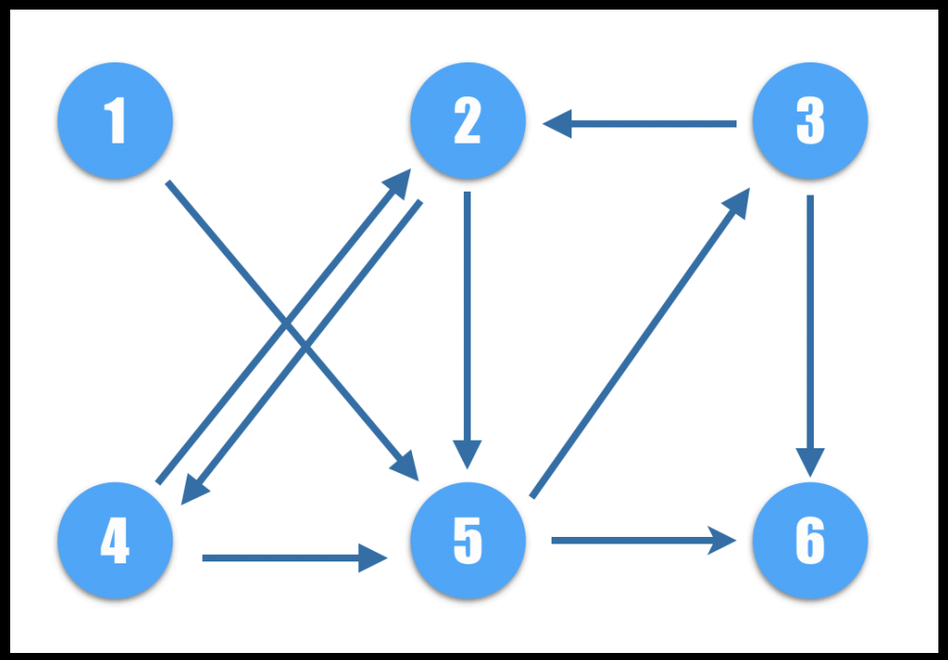
또다른 형태의 예제를 살펴보겠습니다.



Examples

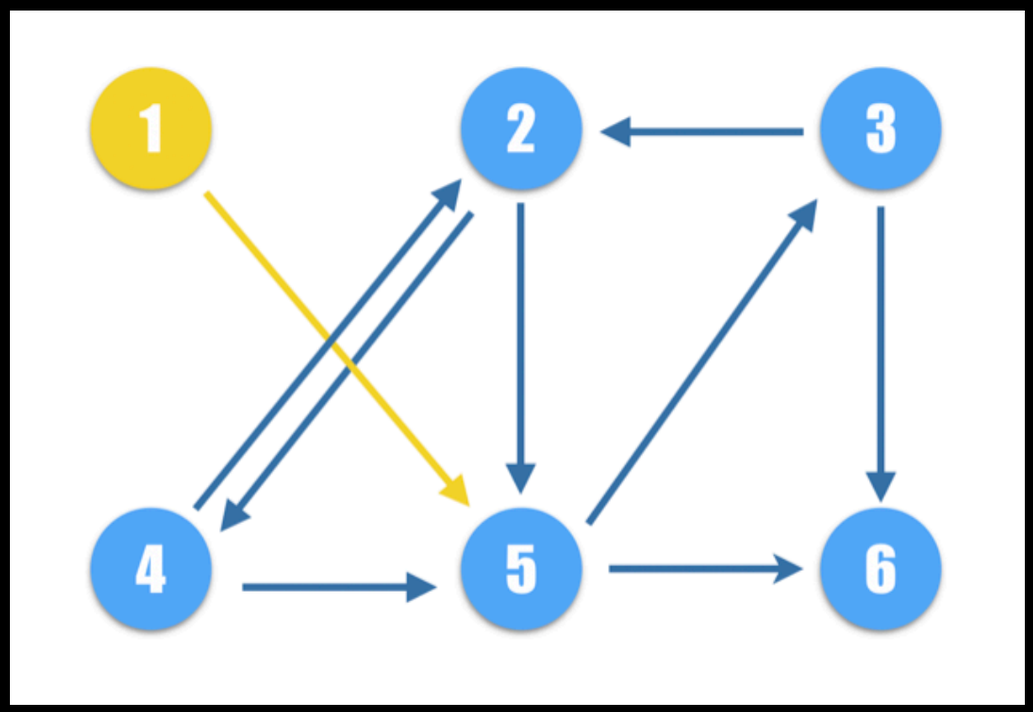
만약 1번 노드에서 출발하여 각 노드들의 최단 루트는 아래와 같습니다.

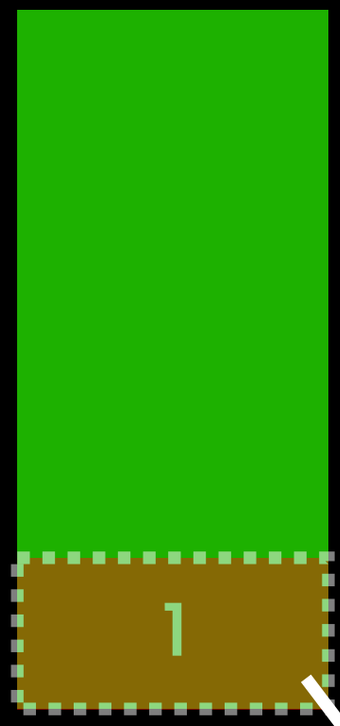
1 -> 5 -> 3 -> 2
1 -> 5 -> 3
1 -> 5 -> 3 -> 2 -> 4
1 -> 5
1 -> 5 -> 6



Examples

1번 노드를 루트로 두는 방향 그래프에서, BFS를 이용해 MST(Minimum Spanning Tree)를 만드는 과정을 거치면 다음과 같은 과정이 진행됩니다.

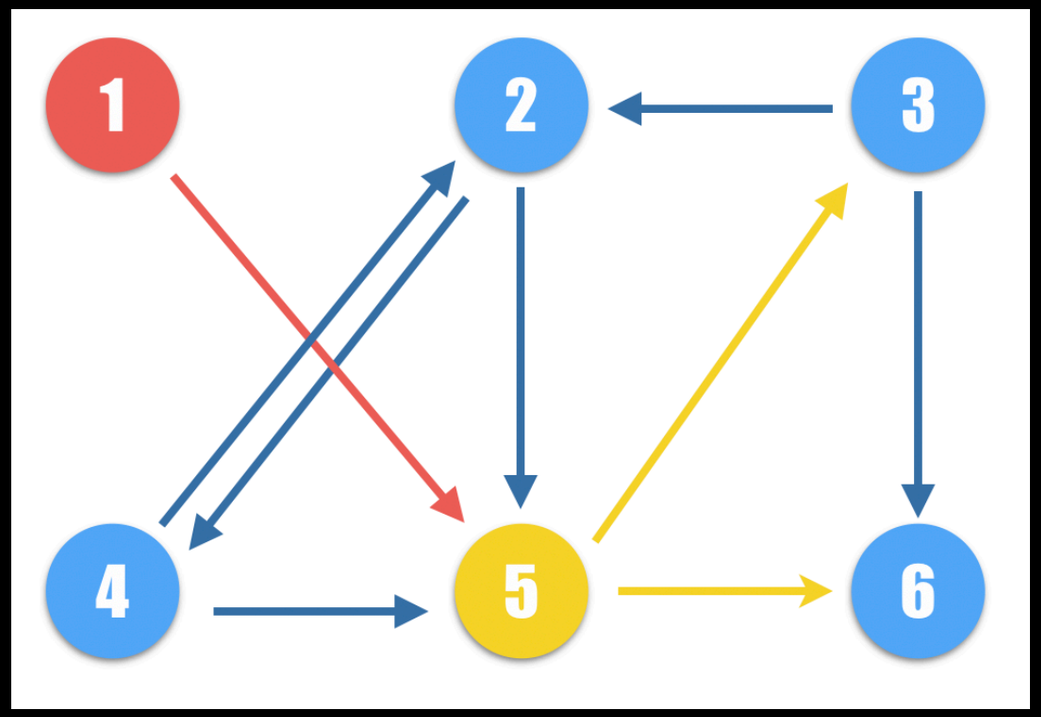


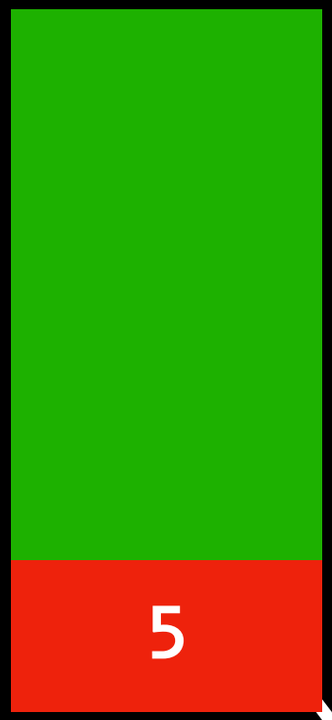


노드별 경로

[0] : [1]
[1] : [1]
[2] : [1]
[3] : [1]
[4] : [1]
[5] : [1]

Examples

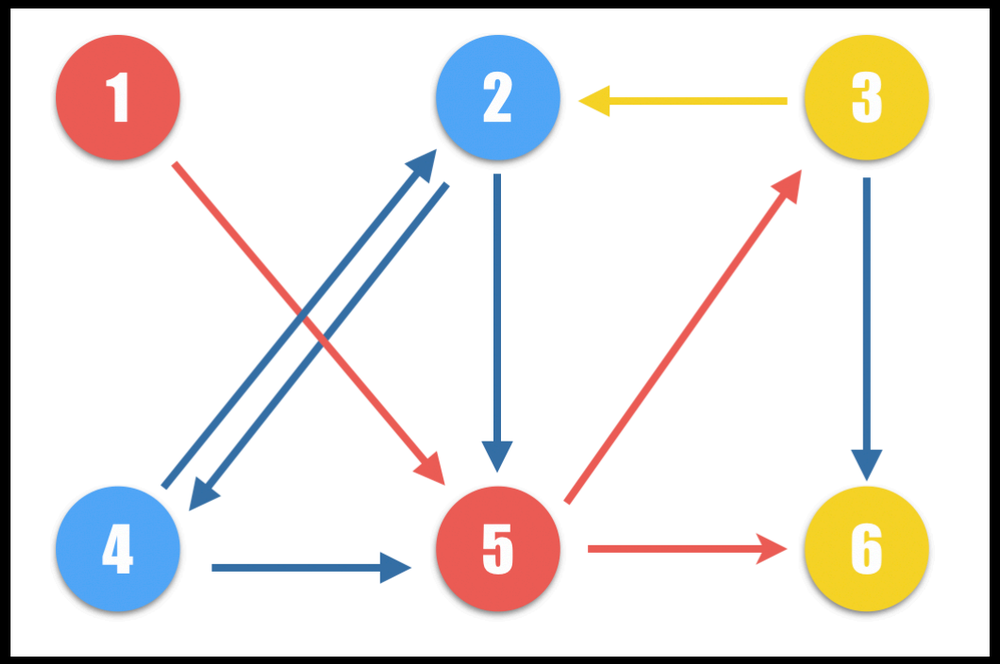


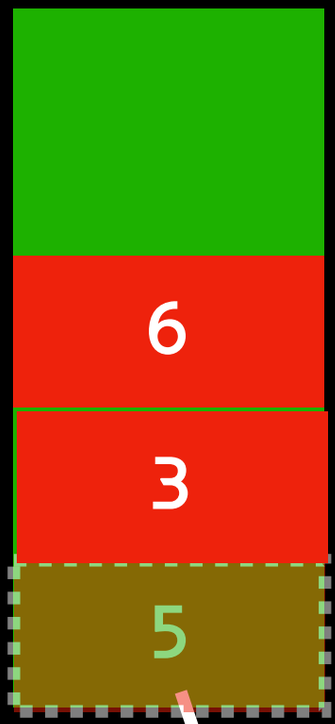


노드별 경로

[0] : [1]
[1] : [1]
[2] : [1]
[3] : [1]
[4] : [1, 5]
[5] : [1]

Examples

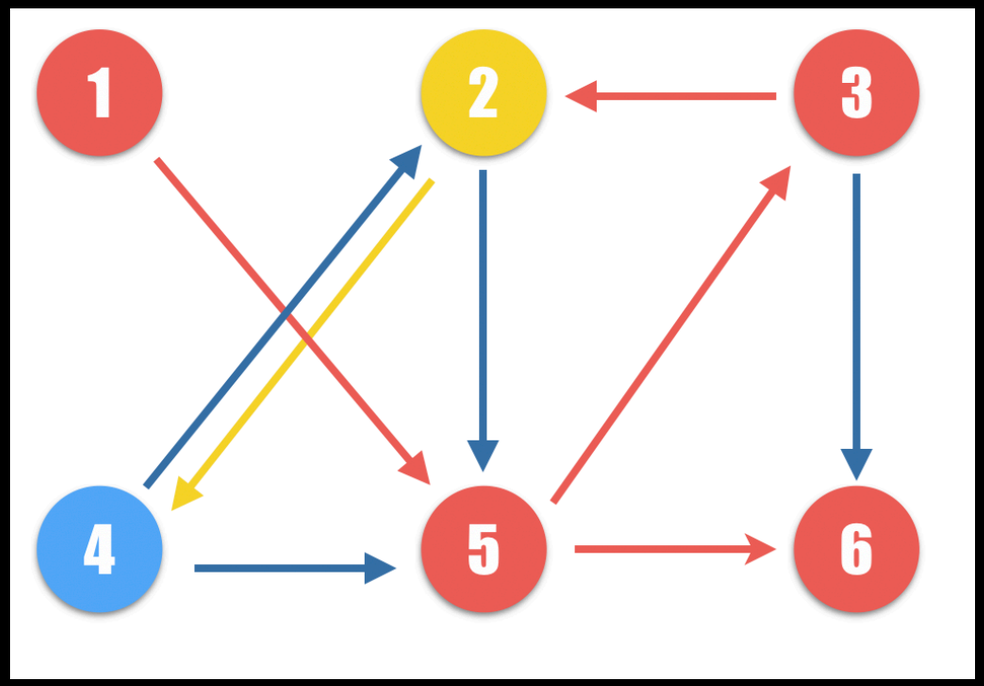


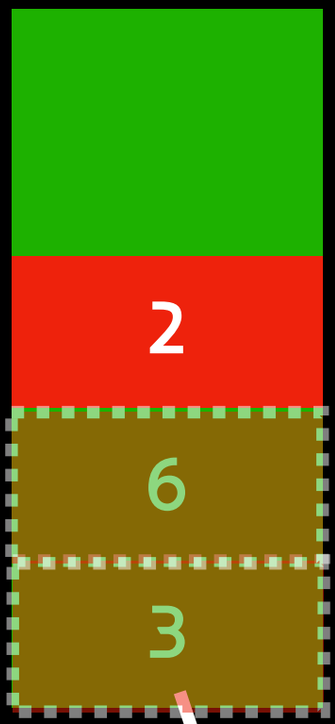


노드별 경로

[0] : [1]
[1] : [1]
[2] : [1, 5, 3]
[3] : [1]
[4] : [1, 5]
[5] : [1, 5, 6]

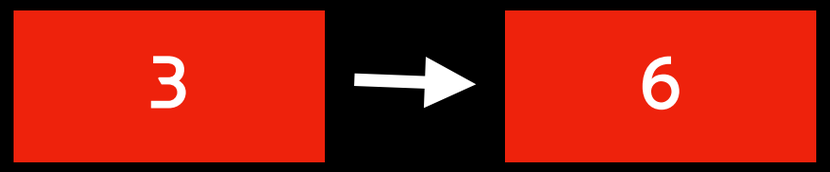
Examples



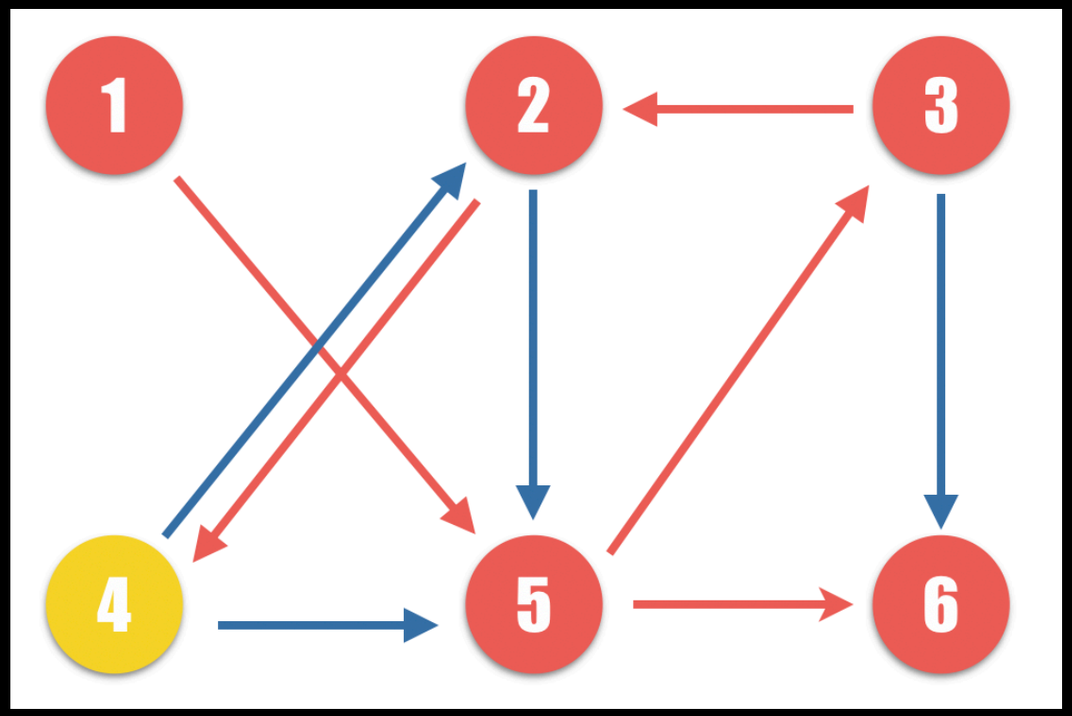


노드별 경로

[0] : [1]
[1] : [1, 5, 3, 2]
[2] : [1, 5, 3]
[3] : [1]
[4] : [1, 5]
[5] : [1, 5, 6]

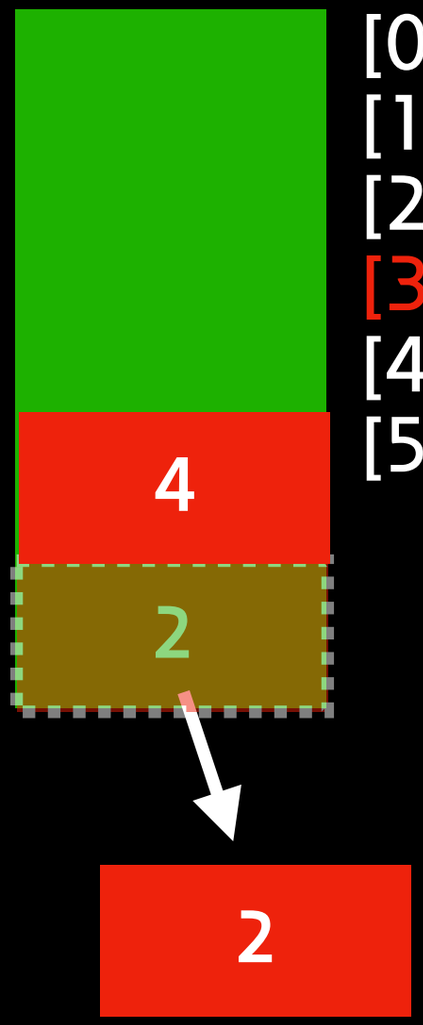


Examples

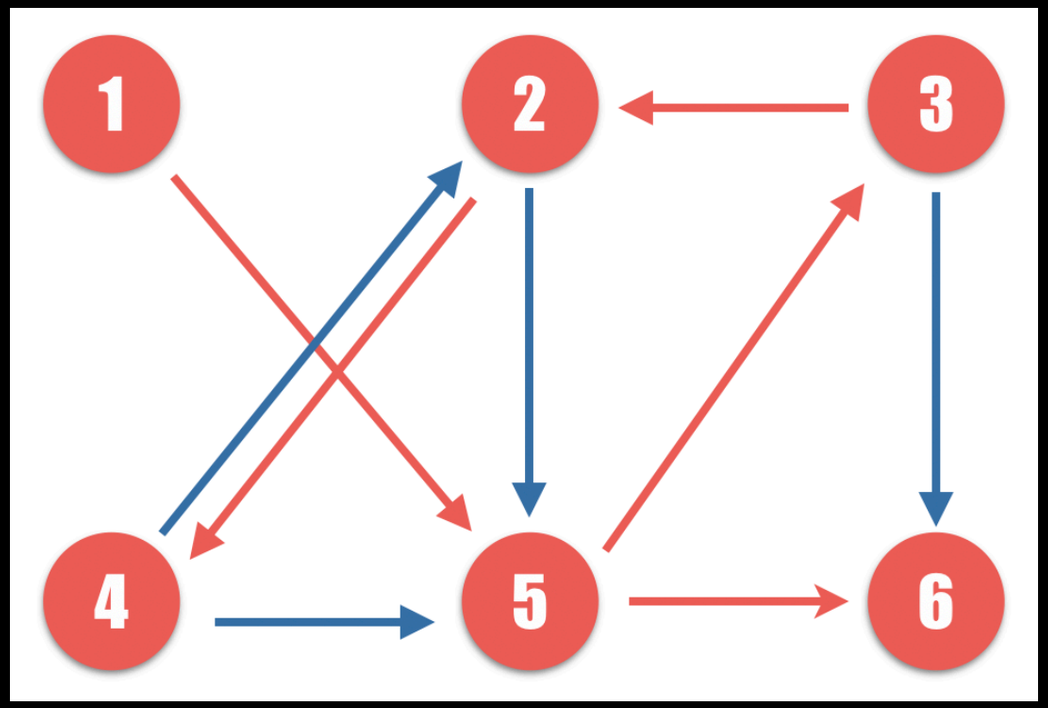


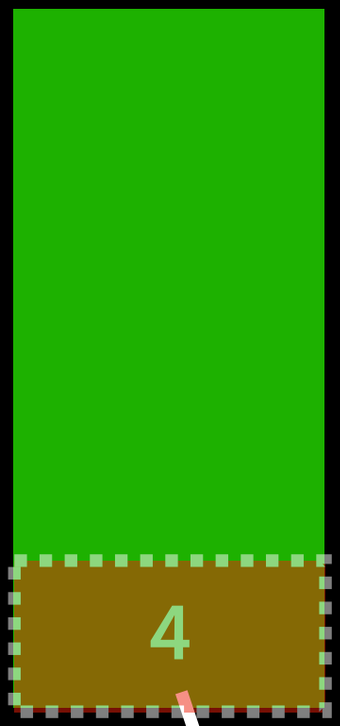
노드별 경로

[0] : [1]
[1] : [1, 5, 3, 2]
[2] : [1, 5, 3]
[3] : [1, 5, 3, 2, 4]
[4] : [1, 5]
[5] : [1, 5, 6]



Examples



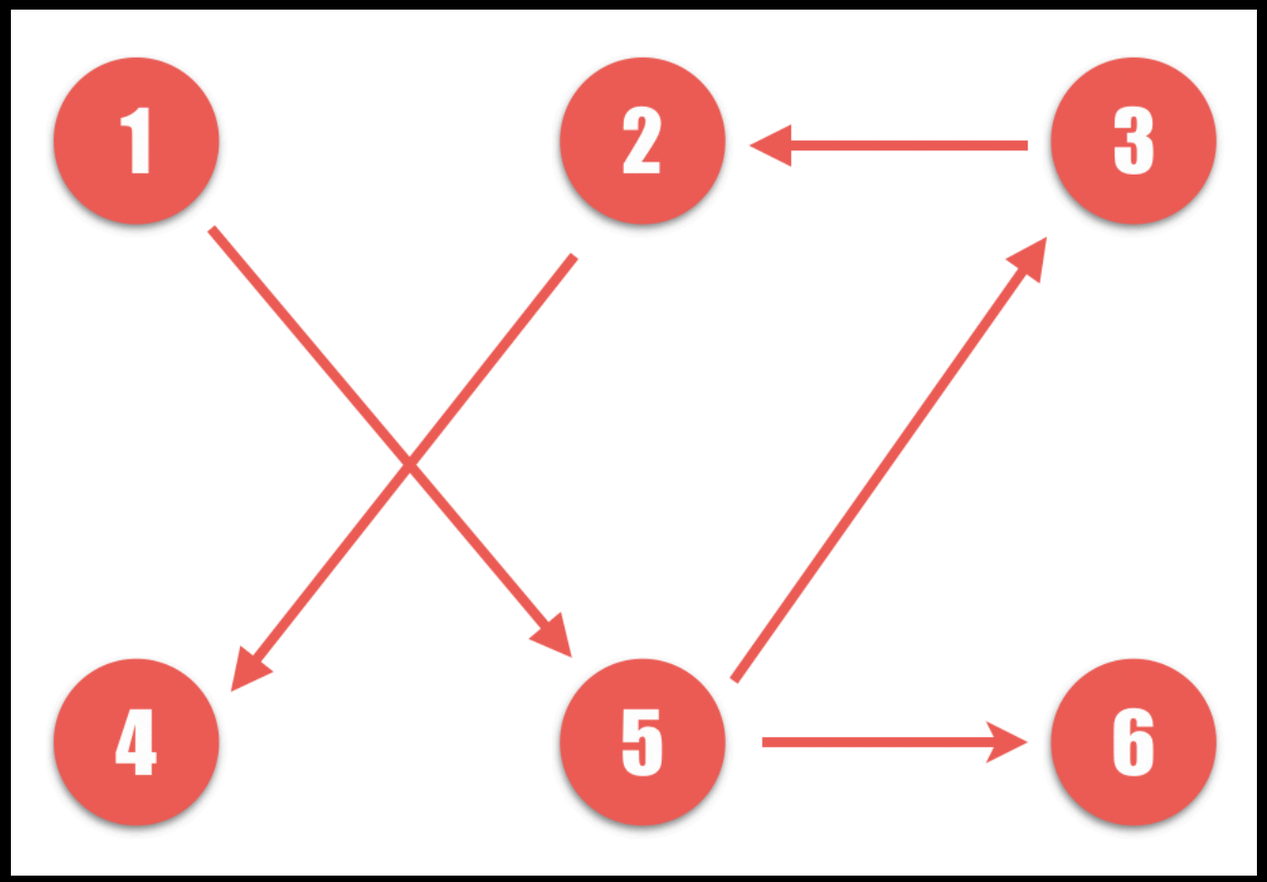


노드별 경로

[0] : [1]
[1] : [1, 5, 3, 2]
[2] : [1, 5, 3]
[3] : [1, 5, 3, 2, 4]
[4] : [1, 5]
[5] : [1, 5, 6]

Examples

모든 노드를 큐에서 넣었다가 빼면 최종 경로는 아래와 같이 됩니다.



최종 노드별 경로

[0] : [1]
[1] : [1, 5, 3, 2]
[2] : [1, 5, 3]
[3] : [1, 5, 3, 2, 4]
[4] : [1, 5]
[5] : [1, 5, 6]

Implementation

Swift를 활용하여 가장 기본적인 위에서 살펴본 2개의 예제를 직접 구현해보겠습니다. 우선 필요한 객체와 메소드는 아래와 같습니다.

필요한 객체

- 정점(Vertex) 객체
- 간선(Edge) 객체

그래프 기본 메소드

- breadthFirstSearch : BFS(너비 우선 탐색)를 실행하는 함수

Implementation

```
public class NodeGraph : CustomStringConvertible, Equatable {
    public var neighbors: [EdgeGraph]

    public private(set) var label: String
    public var distance: Int?
    public var visited: Bool

    public init(_ label: String) {
        self.label = label
        neighbors = []
        visited = false
    }

    public var description: String {
        if let distance = distance {
          return "Node(label: \(label), distance: \(distance))"
        }

        return "Node(label: \(label), distance: infinity)"
    }

    public var hasDistance: Bool {
        return distance != nil
    }

    public func remove(_ edge: EdgeGraph) {
        neighbors.remove(at: neighbors.index { $0 === edge }!)
    }

    static public func == (_ lhs: NodeGraph, rhs: NodeGraph) -> Bool {
        return lhs.label == rhs.label && lhs.neighbors == rhs.neighbors
    }
}
```

Implementation

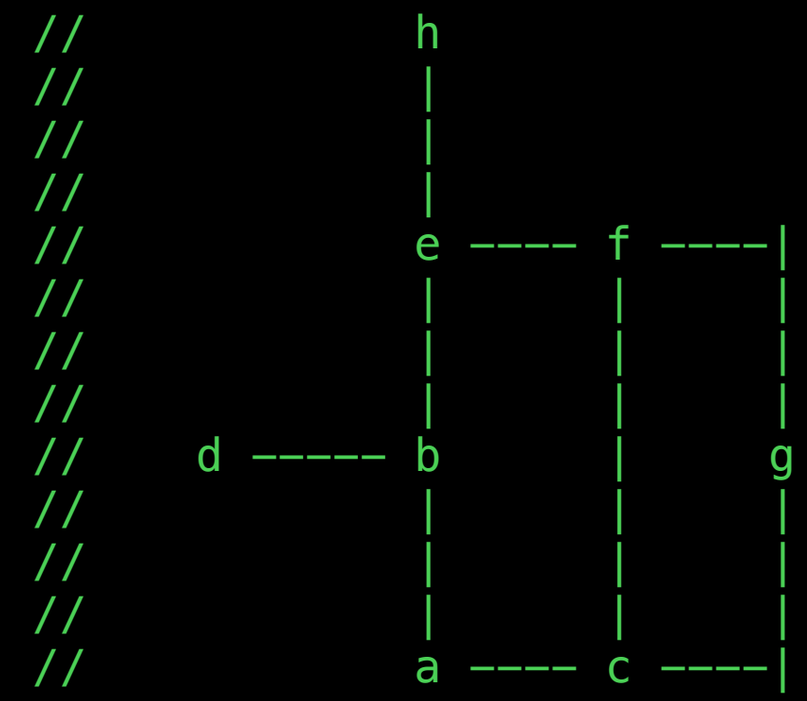
```
public class EdgeGraph : Equatable {
    public var neighbor: NodeGraph

    public init(_ neighbor: NodeGraph) {
        self.neighbor = neighbor
    }

    static public func == (_ lhs: EdgeGraph, rhs: EdgeGraph) -> Bool {
        return lhs.neighbor == rhs.neighbor
    }
}
```

Implementation

```swift
func breadthFirstSearch(_ graph: Graph, source: NodeGraph) -> [String] {
    let queue = Queue<NodeGraph>()

    // 최초 시작 위치를 Queue 에 담습니다.
    queue.enqueue(source)
    print("Enqueue(+) : \(source.label)")

    var nodesExplored = [source.label]
    // 최초 노드를 방문한 것으로 표시합니다.
    source.visited = true

    while let current = queue.dequeue() {
        print("Dequeue(-) : \(current.label)")
        for edge in current.neighbors {
            let neighborNode = edge.neighbor
            if !neighborNode.visited {
                queue.enqueue(neighborNode)
                print("Enqueue(+) : \(neighborNode.label)")
                neighborNode.visited = true
                nodesExplored.append(neighborNode.label)
            }
        }
    }

    return nodesExplored
}
```

Implementation

```
let graph = Graph()

let nodeA = graph.addNode("a")
let nodeB = graph.addNode("b")
let nodeC = graph.addNode("c")
let nodeD = graph.addNode("d")
let nodeE = graph.addNode("e")
let nodeF = graph.addNode("f")
let nodeG = graph.addNode("g")
let nodeH = graph.addNode("h")

graph.addEdge(nodeA, neighbor: nodeB)
graph.addEdge(nodeA, neighbor: nodeC)
graph.addEdge(nodeB, neighbor: nodeD)
graph.addEdge(nodeB, neighbor: nodeE)
graph.addEdge(nodeC, neighbor: nodeF)
graph.addEdge(nodeC, neighbor: nodeG)
graph.addEdge(nodeE, neighbor: nodeH)
graph.addEdge(nodeE, neighbor: nodeF)
graph.addEdge(nodeF, neighbor: nodeG)
```



Implementation

```
// Enqueue(+) : a - Queue [ a ]
//
// Dequeue(-) : a - Queue [ ] => a
// Enqueue(+) : b - Queue [ b ]
// Enqueue(+) : c - Queue [ b, c ]
// Dequeue(-) : b - Queue [ c ] => a, b
// Enqueue(+) : d - Queue [ c, d ]
// Enqueue(+) : e - Queue [ c, d, e ]
// Dequeue(-) : c - Queue [ d, e ] => a, b, c
// Enqueue(+) : f - Queue [ d, e, f ]
// Enqueue(+) : g - Queue [ d, e, f, g ]
// Dequeue(-) : d - Queue [ e, f, g ] => a, b, c, d
// Dequeue(-) : e - Queue [ f, g ] => a, b, c, d, e
// Enqueue(+) : h - Queue [ f, g, h ]
// Dequeue(-) : f - Queue [ g, h ] => a, b, c, d, e, f
// Dequeue(-) : g - Queue [ h ] => a, b, c, d, e, f, g
// Dequeue(-) : h - Queue [ ] => a, b, c, d, e, f, g, h

print(nodesExplored)
// ["a", "b", "c", "d", "e", "f", "g", "h"]
```

Implementation

이번에는 두번째 예제에 대한 구현 소스를 살펴보겠습니다.

```
class NodeGraphPath<T>
{
    let value: T
    var edges = [EdgeGraphPath<T>]()
    var visited = false

    init(value: T) {
        self.value = value
    }

    func appendEdgeTo(_ node: NodeGraphPath<T>) {
        let edge = EdgeGraphPath<T>(from: self, to: node)
        self.edges.append(edge)
    }
}
```

Implementation

```
class EdgeGraphPath<T> {
    weak var source: NodeGraphPath<T>?
    let destination: NodeGraphPath<T>

    init(from source: NodeGraphPath<T>, to destination: NodeGraphPath<T>) {
        self.source = source
        self.destination = destination
    }
}
```

Implementation

```swift
func breadthFirstSearch(n: Int, edges: [(Int, Int)]) -> [Any] {
    let nodes = (0..<n).map({ NodeGraphPath<Int>(value: $0 + 1) })

    for (from, to) in edges {
        nodes[from - 1].appendEdgeTo(nodes[to - 1])
    }

    var shortest = Array(repeating: [1], count: n)

    let queue = Queue<NodeGraphPath<Int>>()

    // 시작 노드를 큐에 삽입 및 방문 상태 체크
    queue.enqueue(nodes[0])
    nodes[0].visited = true

    // 큐에서 최상단 노드를 하나씩 빼면서 큐가 비어 있을 때까지 반복
    while let node = queue.dequeue() {
        for edge in node.edges {
            let dest = edge.destination

            // 현재 대상 노드를 큐에 삽입 및 방문 상태 체크
            guard dest.visited == false else { continue }
            queue.enqueue(dest)
            dest.visited = true

            // 현재 노드에 대한 총 이동 경로 노드를 저장
            shortest[dest.value - 1] = shortest[node.value - 1] + [dest.value]
        }
    }

    return shortest
}
```

Implementation

```
print(breadthFirstSearch(n: 6, edges: [(1,5), (2,4), (2,5), (3,2), (3,6), (4,2),
(4,5), (5,3), (5,6)]))

// [[1]
//  [1, 5, 3, 2],
//  [1, 5, 3],
//  [1, 5, 3, 2, 4],
//  [1, 5],
//  [1, 5, 6] ]
```

References

[1] Swift로 그래프 탐색 알고리즘을 실전 문제에 적용해보기 - BFS 편 : https://wlaxhrl.tistory.com/89

[2] DFS (Depth-First Search) BFS (Breadth-First Search) 개념 : https://hucet.tistory.com/83

[3] [알고리즘] DFS & DFS : https://hyesunzzang.tistory.com/186

[4] 너비 우선 탐색 : https://ko.wikipedia.org/wiki/너비_우선_탐색

[5] [Data Structure] 그래프 순회, 탐색(DFS) - 자료 구조 : https://palpit.tistory.com/898

References

[6] DFS (Depth-First Search) BFS (Breadth-First Search) 개념 : https://hucet.tistory.com/83

[7] Understanding Depth & Breadth-First Search in Swift : https://medium.com/swift-algorithms-data-structures/understanding-depth-breadth-first-search-in-swift-90573fd63a36

[8] Breadth first search : https://www.programiz.com/dsa/graph-bfs

[9] [알고리즘] BFS & DFS : https://hyesunzzang.tistory.com/186

[10] 자료구조 :: 그래프(2) "탐색, 깊이우선, 너비우선 탐색” : http://egloos.zum.com/printf/v/755736

Thank you!