SWIFT
Data Structure - Graph(DFS)

Bill Kim(김정훈) | ibillkim@gmail.com

목차

DFS

DFS(Depth-First Search)란 깊이 우선 탐색으로서 그래프에서 모든 경로를 탐색하는데 사용하는 알고리즘입니다.

가중치를 가지지 않는 (무)방향 그래프에서 모든 경로의 경우를 구해볼 때 많이 사용하는 방식입니다.

DFS의 기본 아이디어는 시작 노드의 인접한 이동 가능한 노드를 선택하여 가다가 더이상 갈 수 있는 길이 없을 경우 다시 돌아와서 다른 노드로 이동하고 결국 더이상 이동할 노드가 없을 경우 모든 탐색을 종료하는 방식입니다.

주로 재귀 호출 내지는 스택을 활용하여 사용합니다.

Concept

DFS의 기본적인 알고리즘 흐름은 다음과 같습니다.

- 재귀 호출 방식

1. 시작 노드를 정하고 방문한 것으로 표시
2. 탐색한 리스트를 위한 배열을 선언
3. 시작 노드의 인접 노드 리스트만큼 반복문을 돌린다.
4. 반복분에서 현재 노드가 방문한 적이 없다면 재귀로 DFS 탐색
5. 재귀 DFS 탐색 후 탐색한 노드 탐색 리스트 배열에 추가
6. 인접 노드가 없으면 탐색 종료

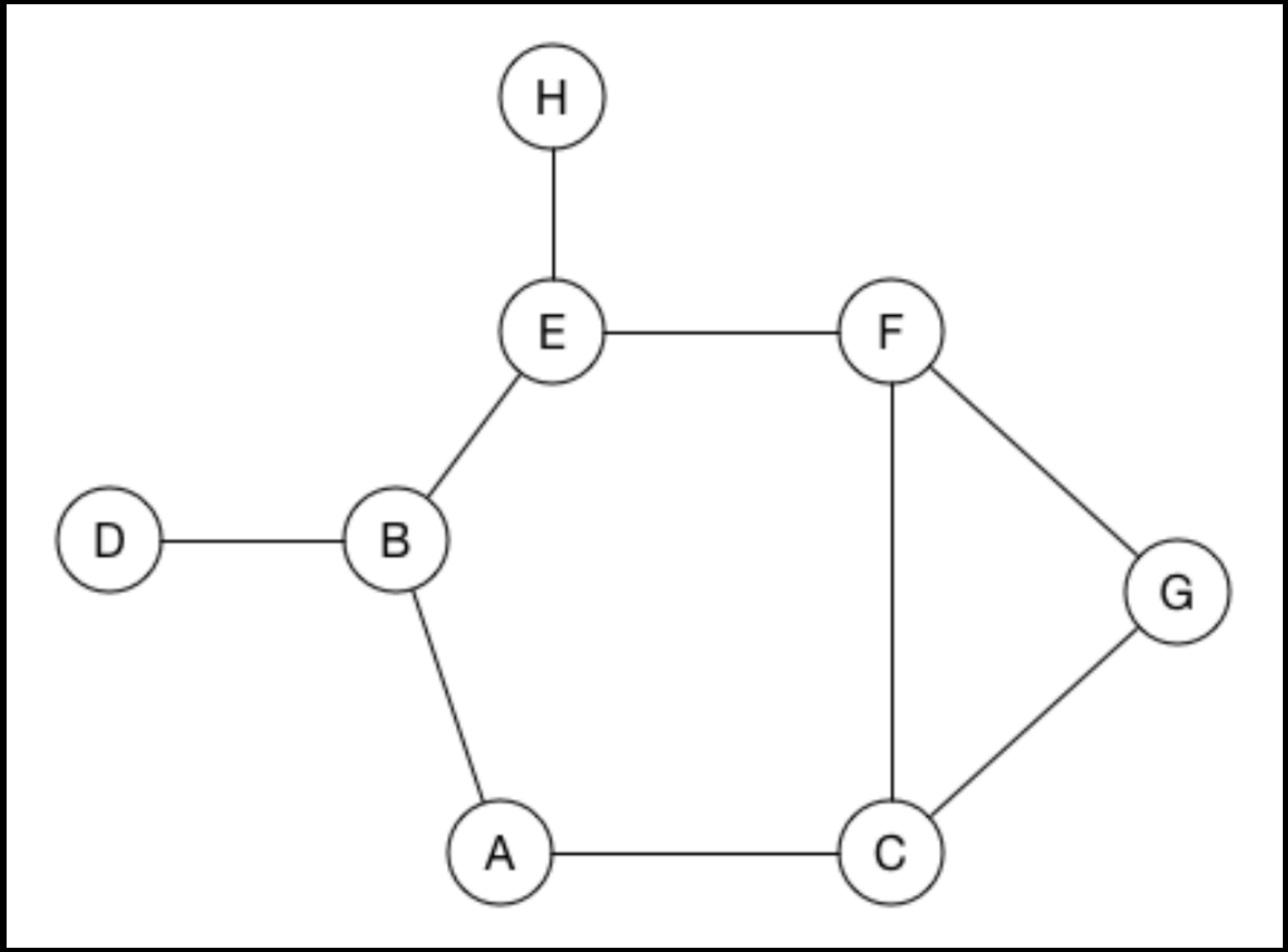
Concept

- 스택 방식

1. 시작 노드를 정하고 스택에 넣는다.(push)
2. 스택에서 노드를 하나 꺼낸다.(pop)
3. 꺼낸 노드와 인접한 노드가 있는지 확인한다.
4. 인접한 노드가 있고 이미 방문한 노드했던 노드가 아니라면 해당 노드를 스택에 넣는다.(push)
5. 만약 3번에서 더이상 인접 노드가 없다면 스택에서 노드를 하나 꺼낸다.(pop)
6. 다시 3번에서 5번 과정을 스택이 비워질 때까지 계속 반복한다.

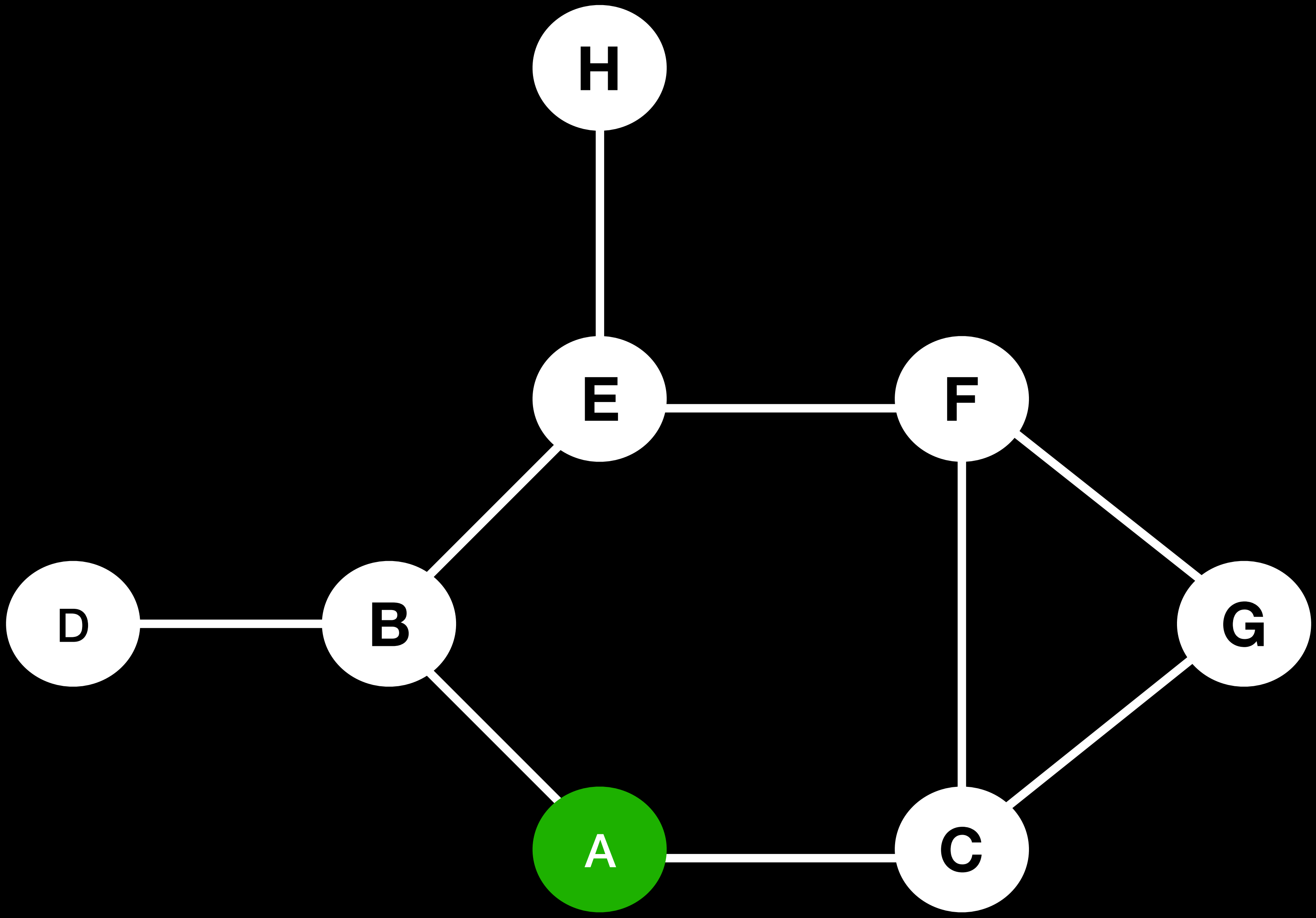
Examples

앞서 설명한 알고리즘을 실제 예제를 통해서 살펴봅니다.

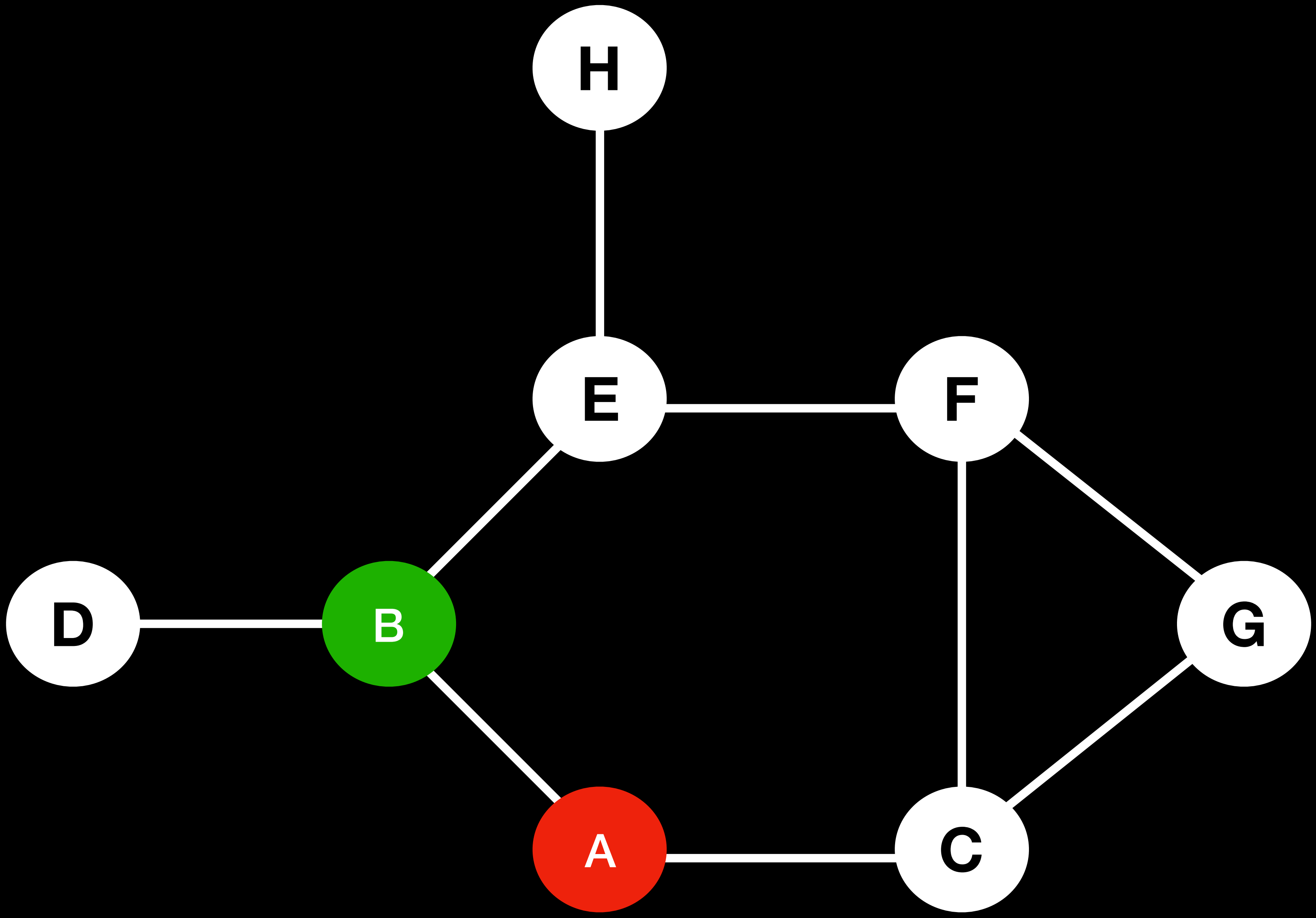


Examples

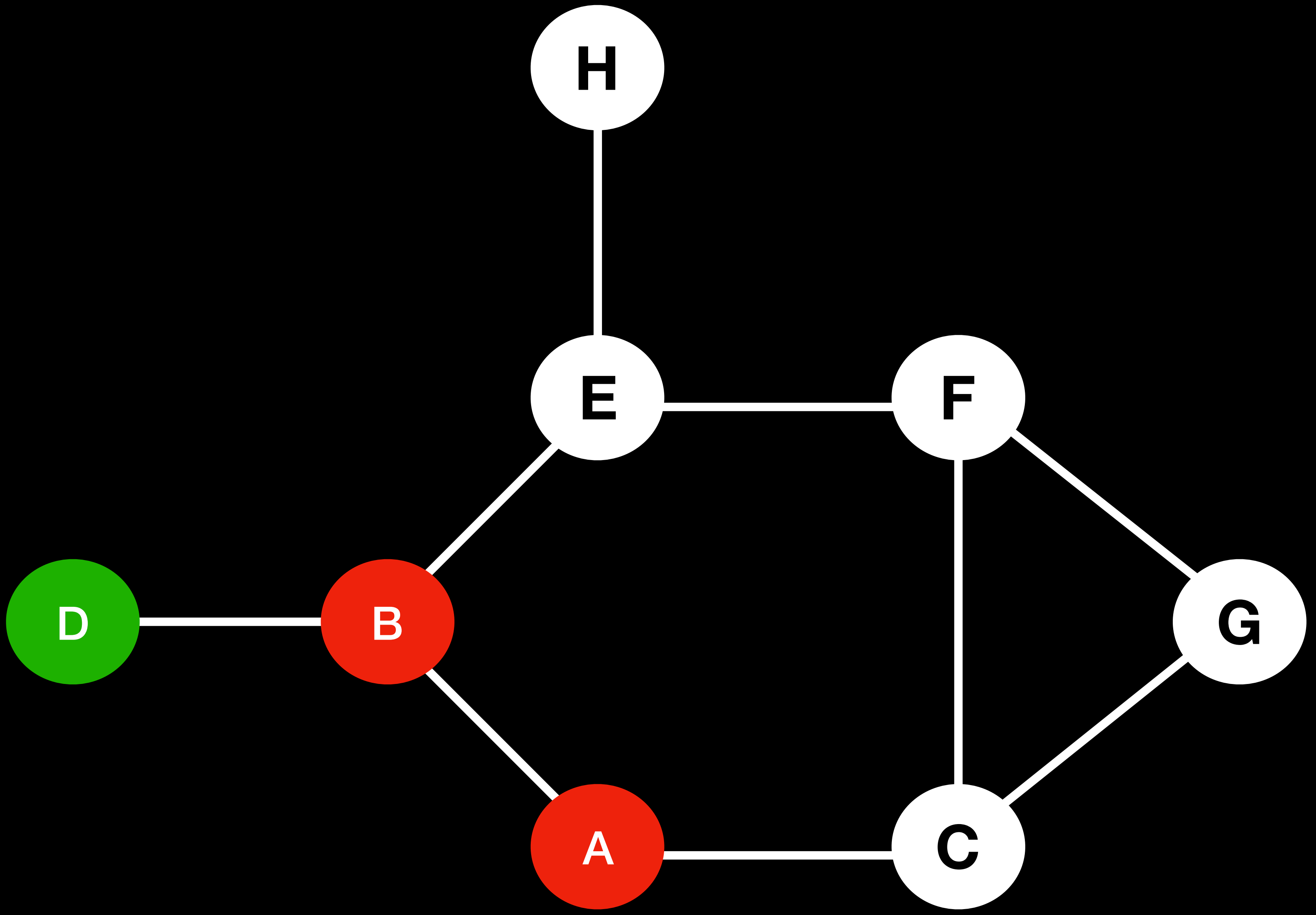
DFS 알고리즘을 통하여 A 노드 부터의 탐색 과정을 살펴보면 다음과 같습니다.



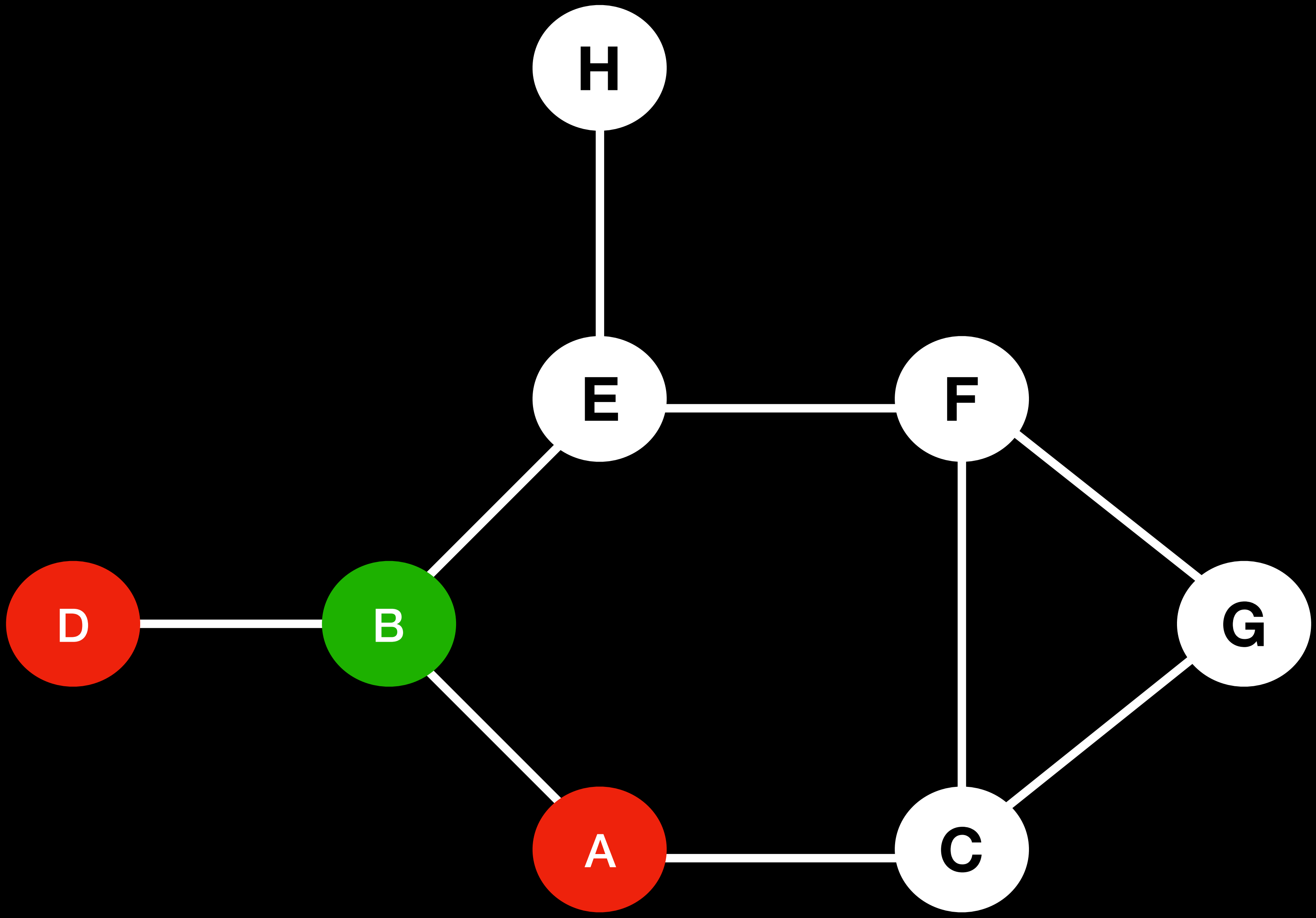
Examples



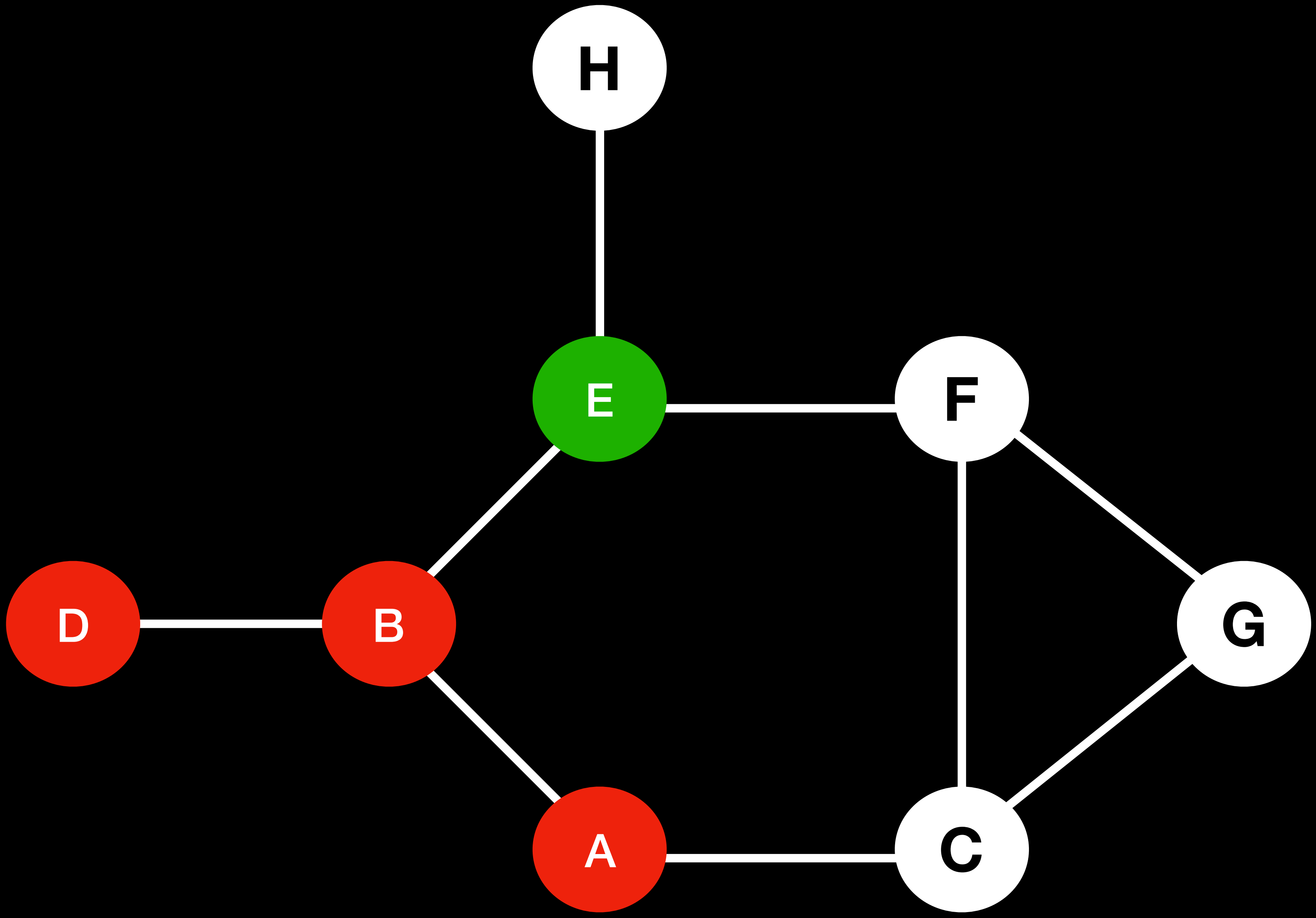
Examples



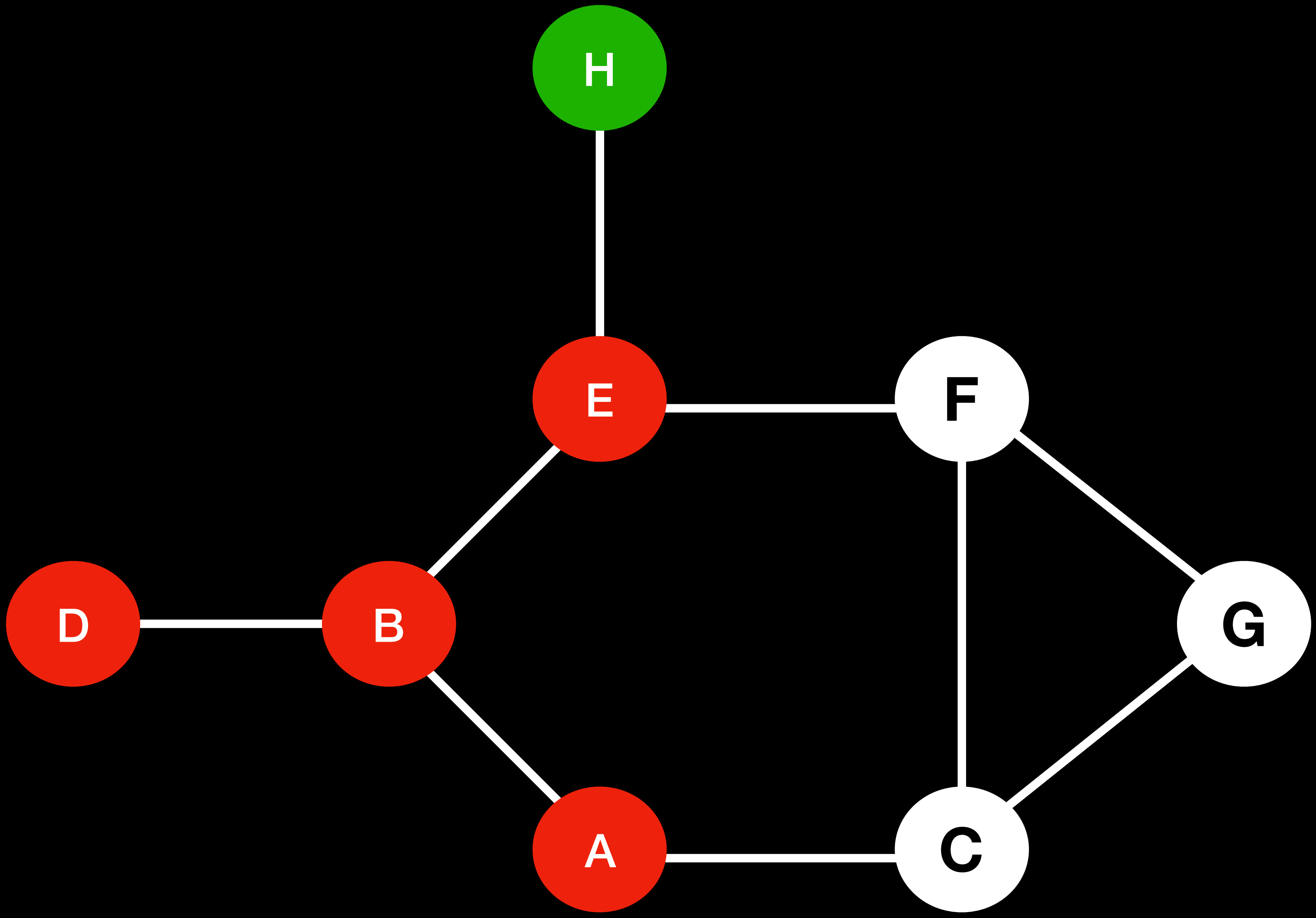
Examples



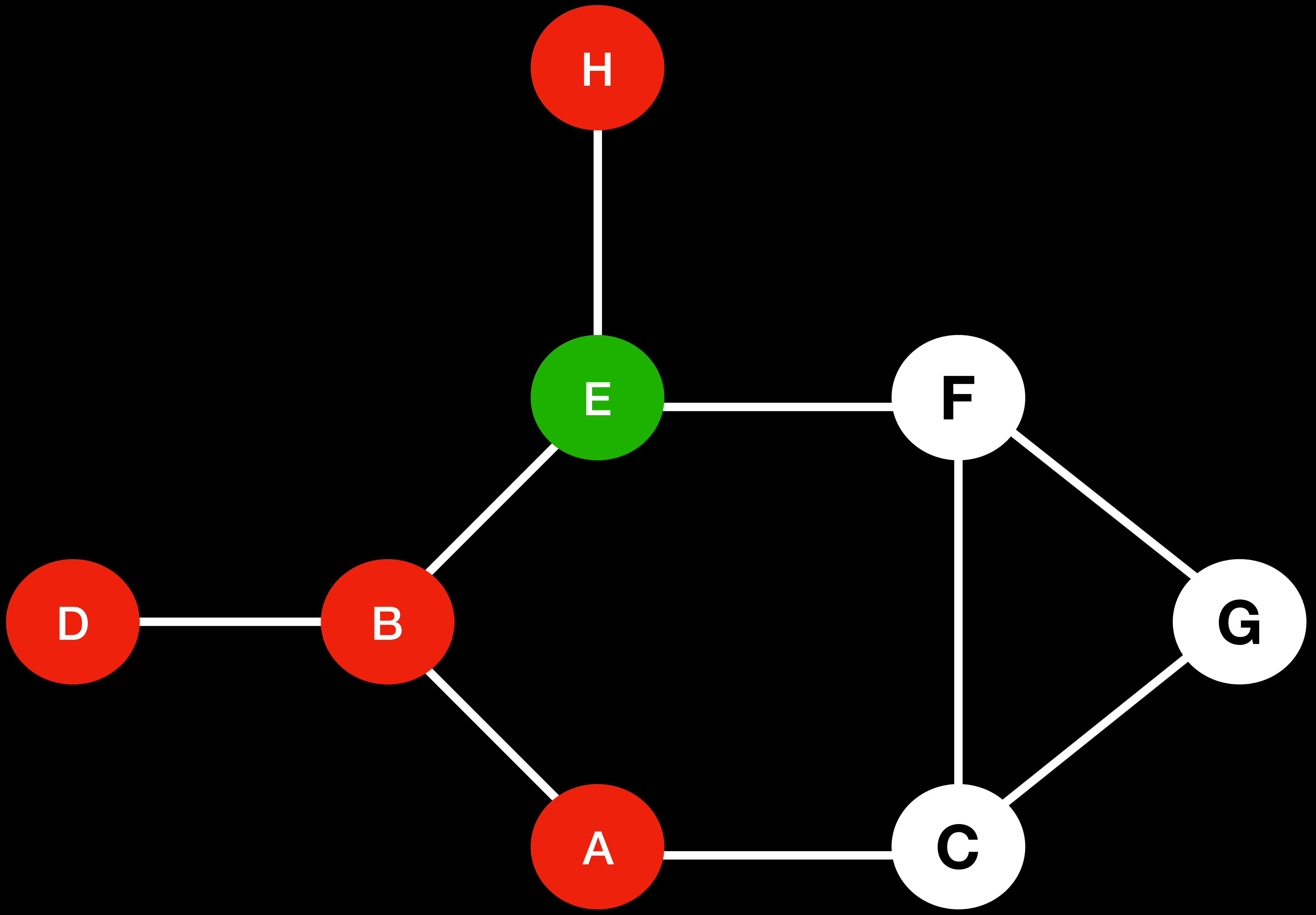
Examples



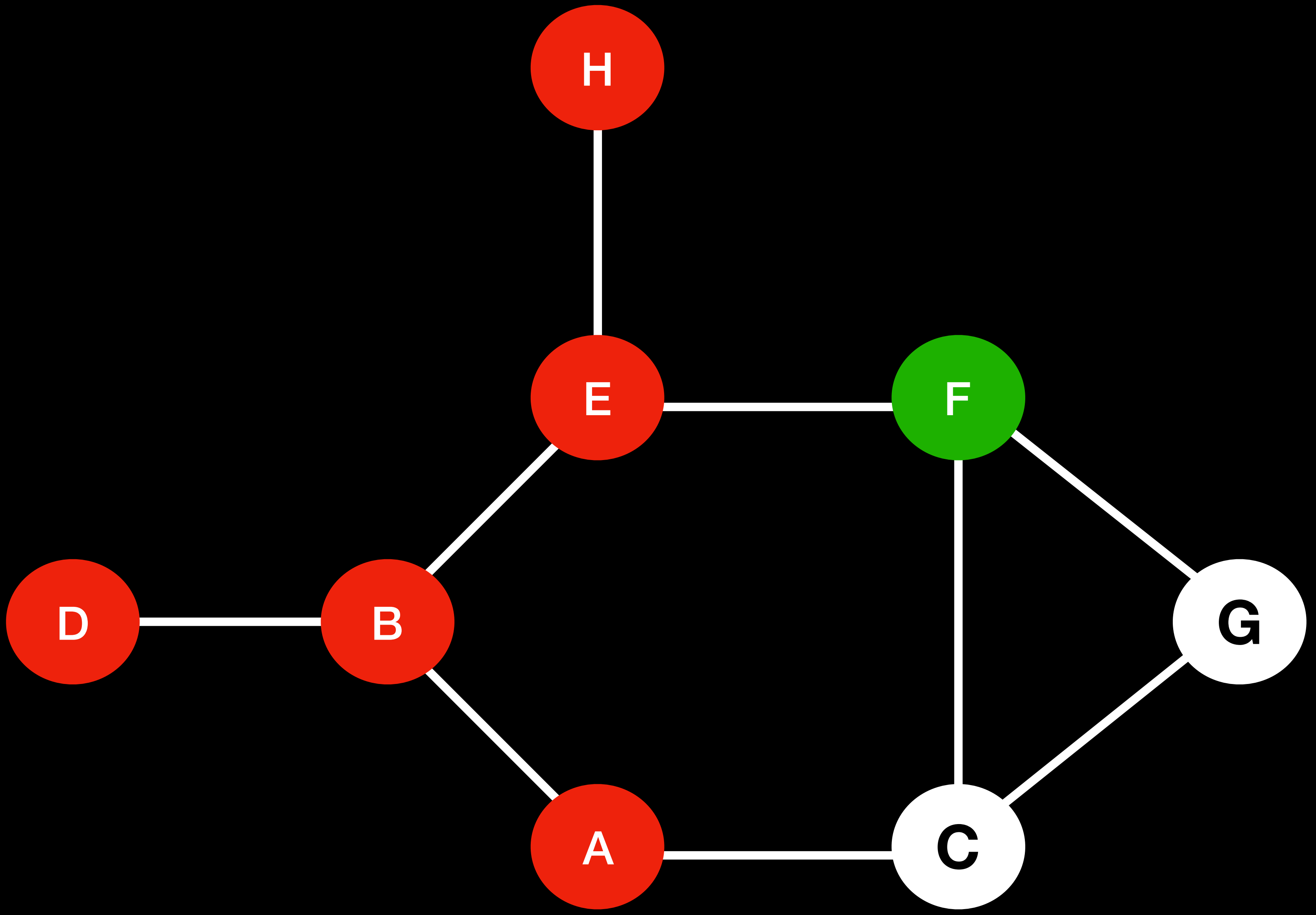
Examples



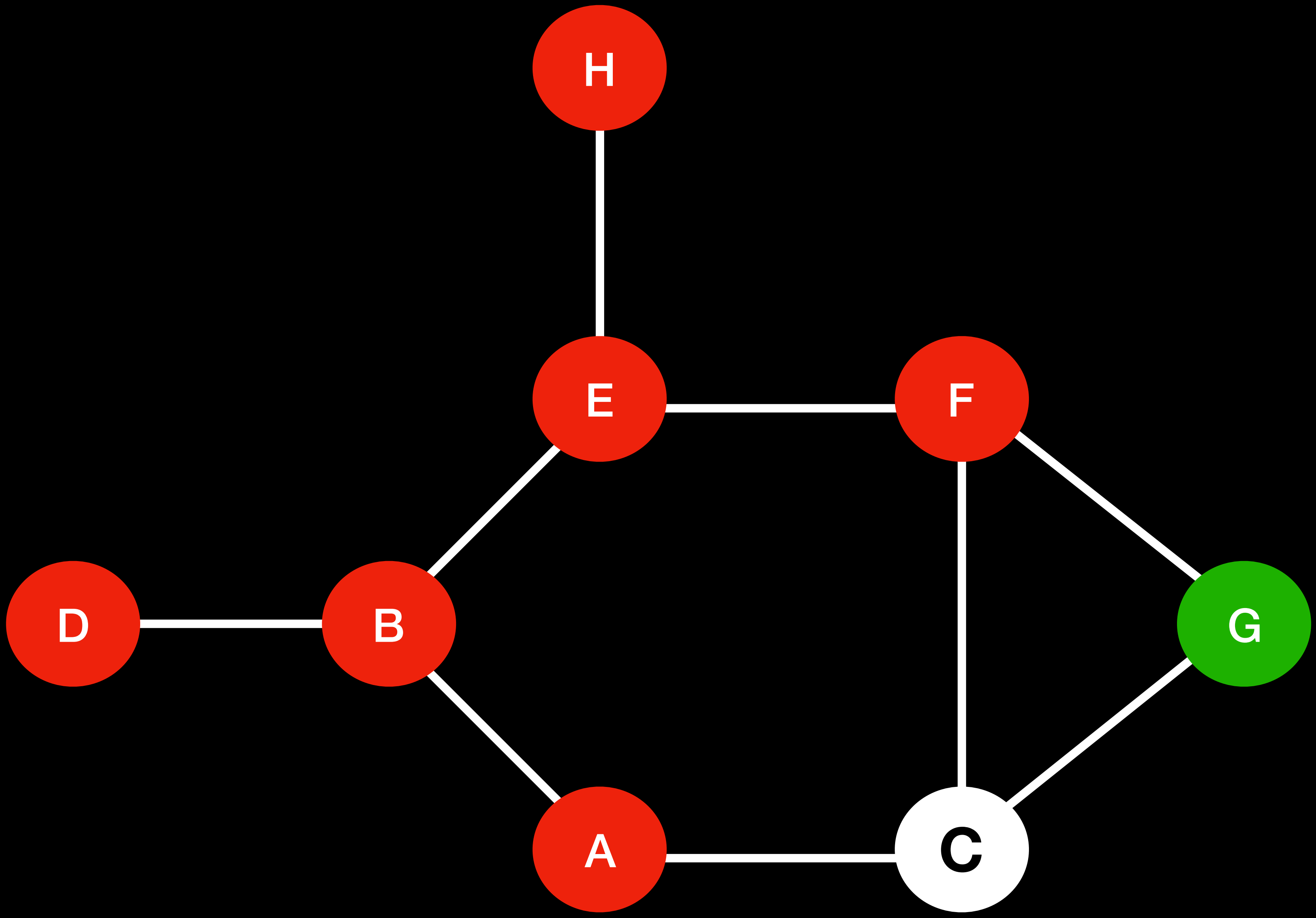
Examples



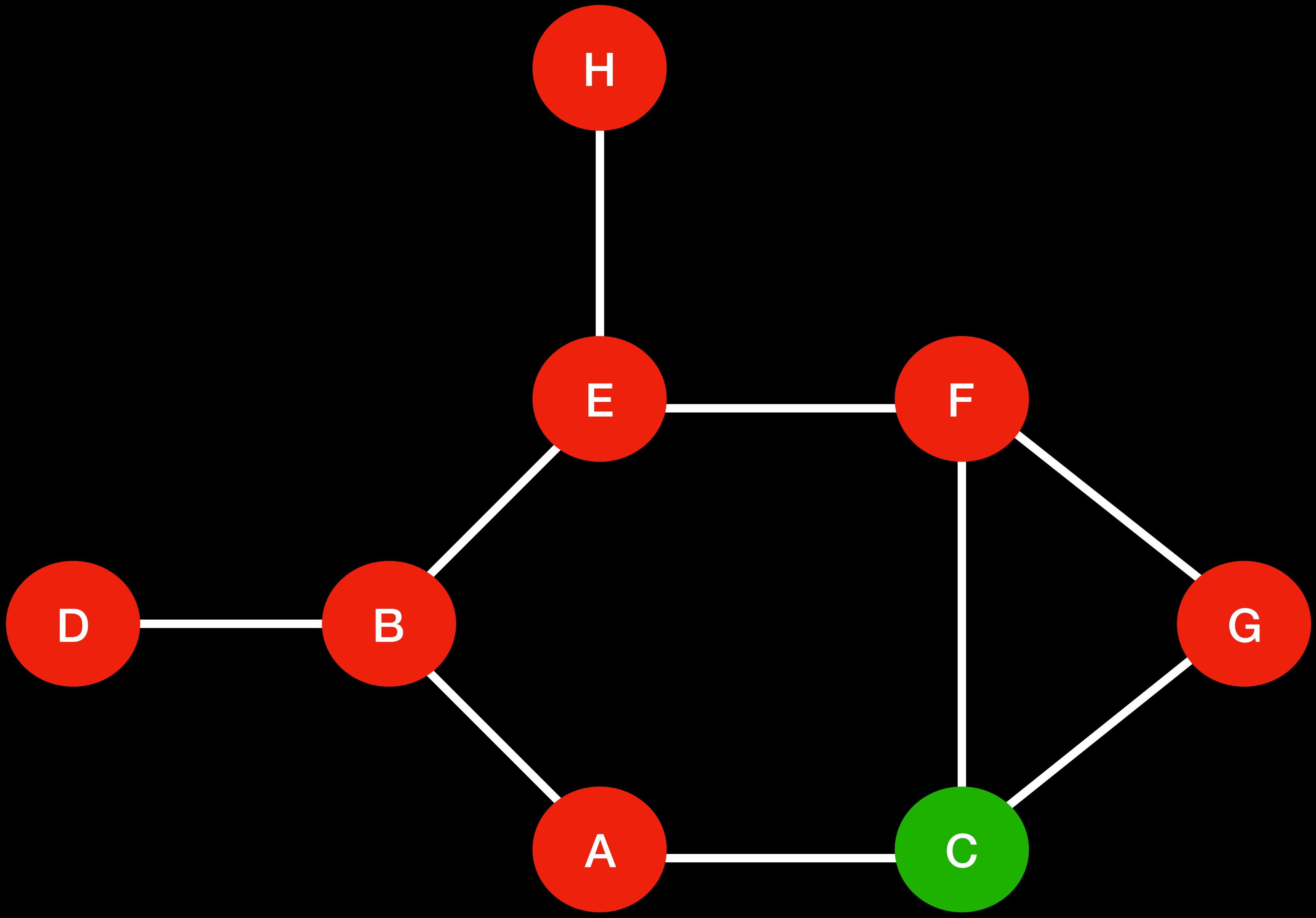
Examples



Examples



Examples



Examples

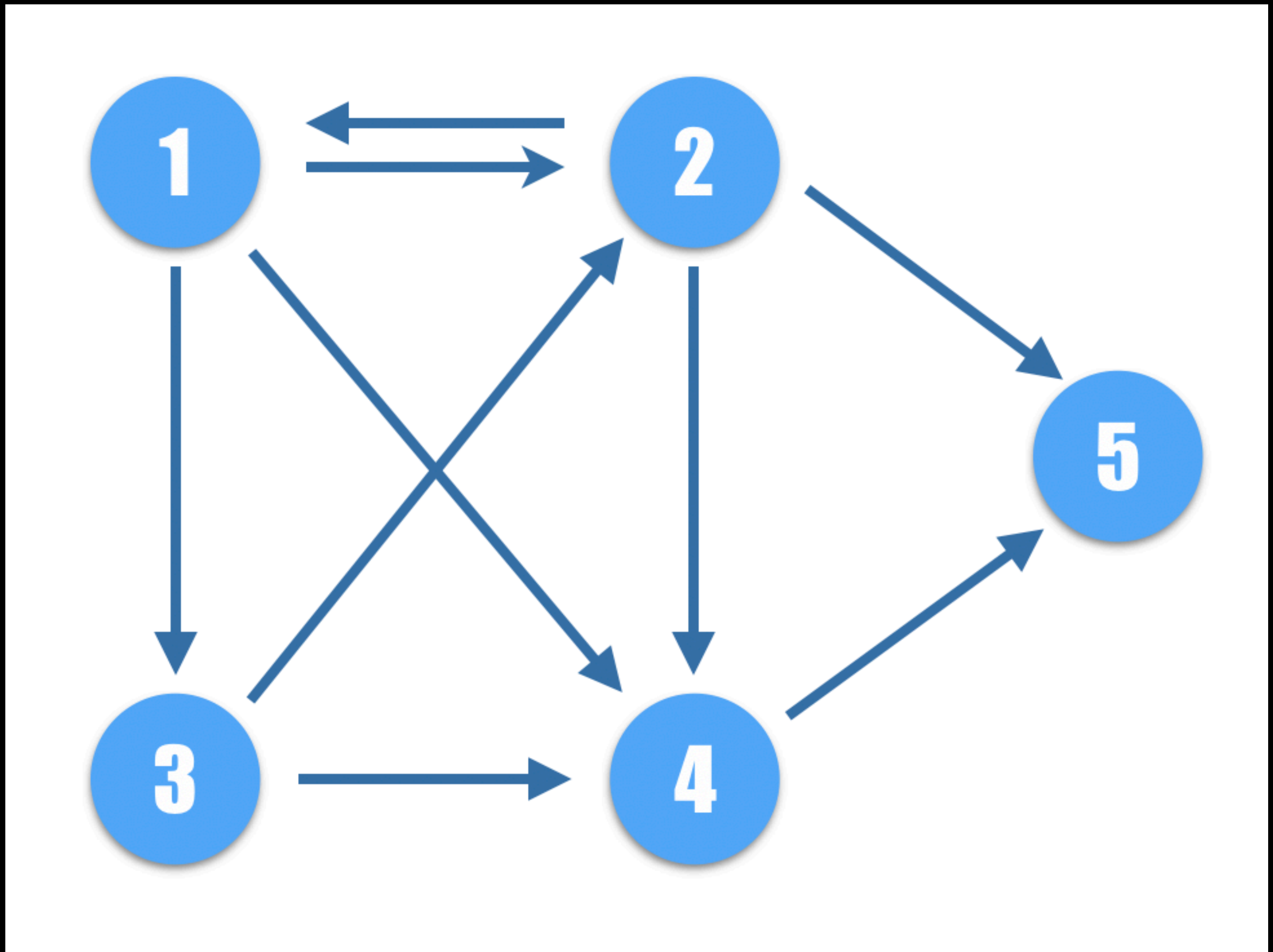
최종 경로를 살펴보면 아래와 같습니다.

```
// node : ["A"]

// neighbors : ["A"]
// node : ["B"]

// neighbors : ["B"]
// node : ["D"]

// neighbors : ["B", "D"]
// node : ["E"]

// neighbors : ["E"]
// node : ["H"]

// neighbors : ["E", "H"]
// node : ["F"]

// neighbors : ["F"]
// node : ["G"]

// neighbors : ["A", "B", "D", "E", "H", "F", "G"]
// node : ["C"]
```

최종 경로 : ["A", "B", "D", "E", "H", "F", "G", "C"]

Examples

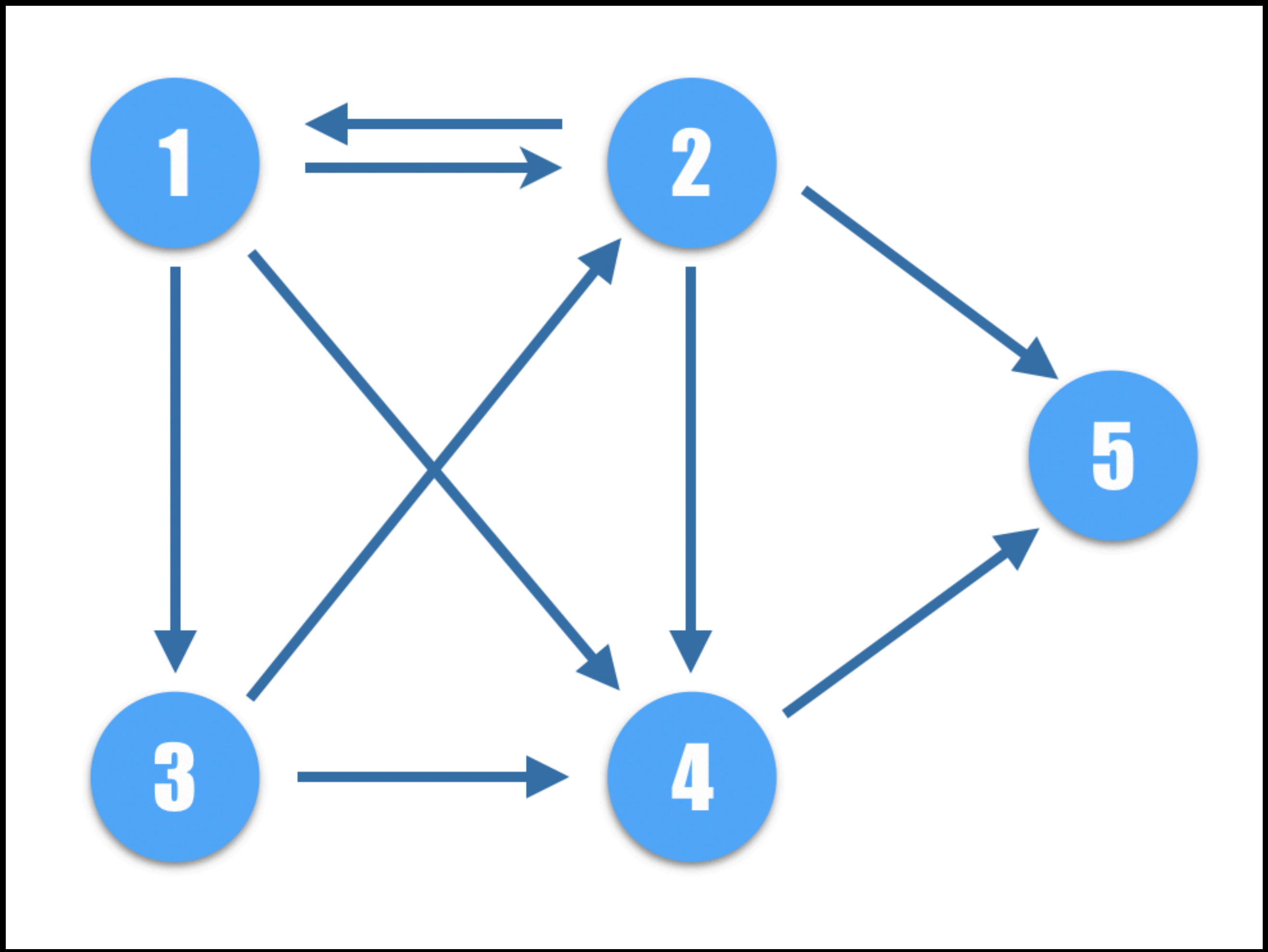
또다른 형태의 예제를 살펴보겠습니다.



Examples

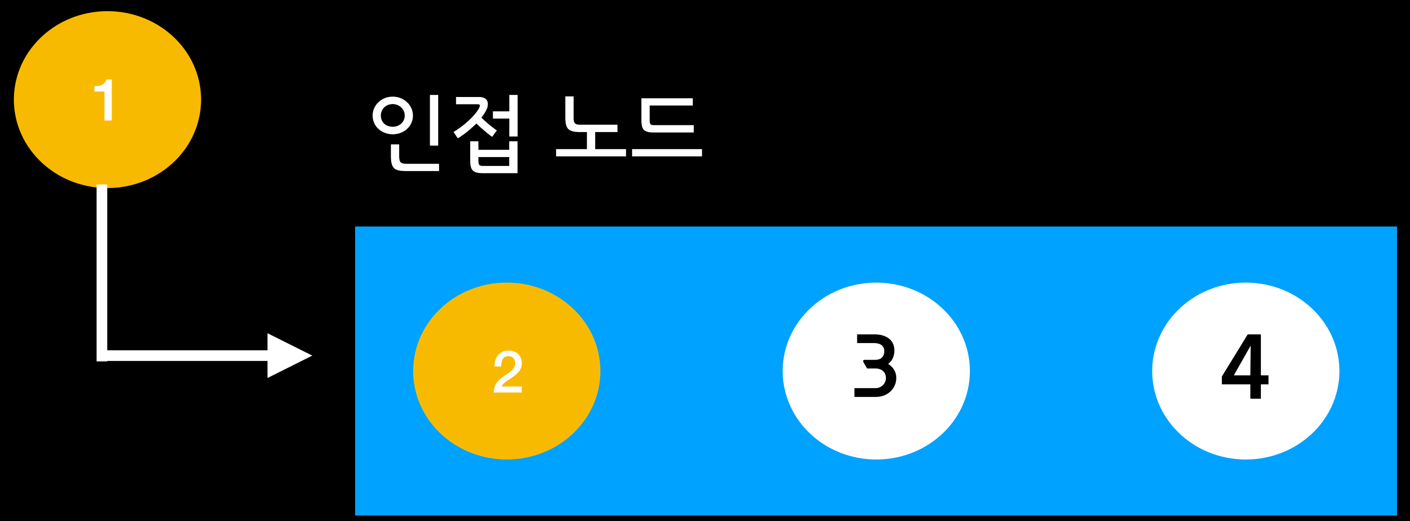
만약 1번 노드에서 출발하여 5번 노드까지 갈 수 있는 경로는 아래와 같습니다.

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



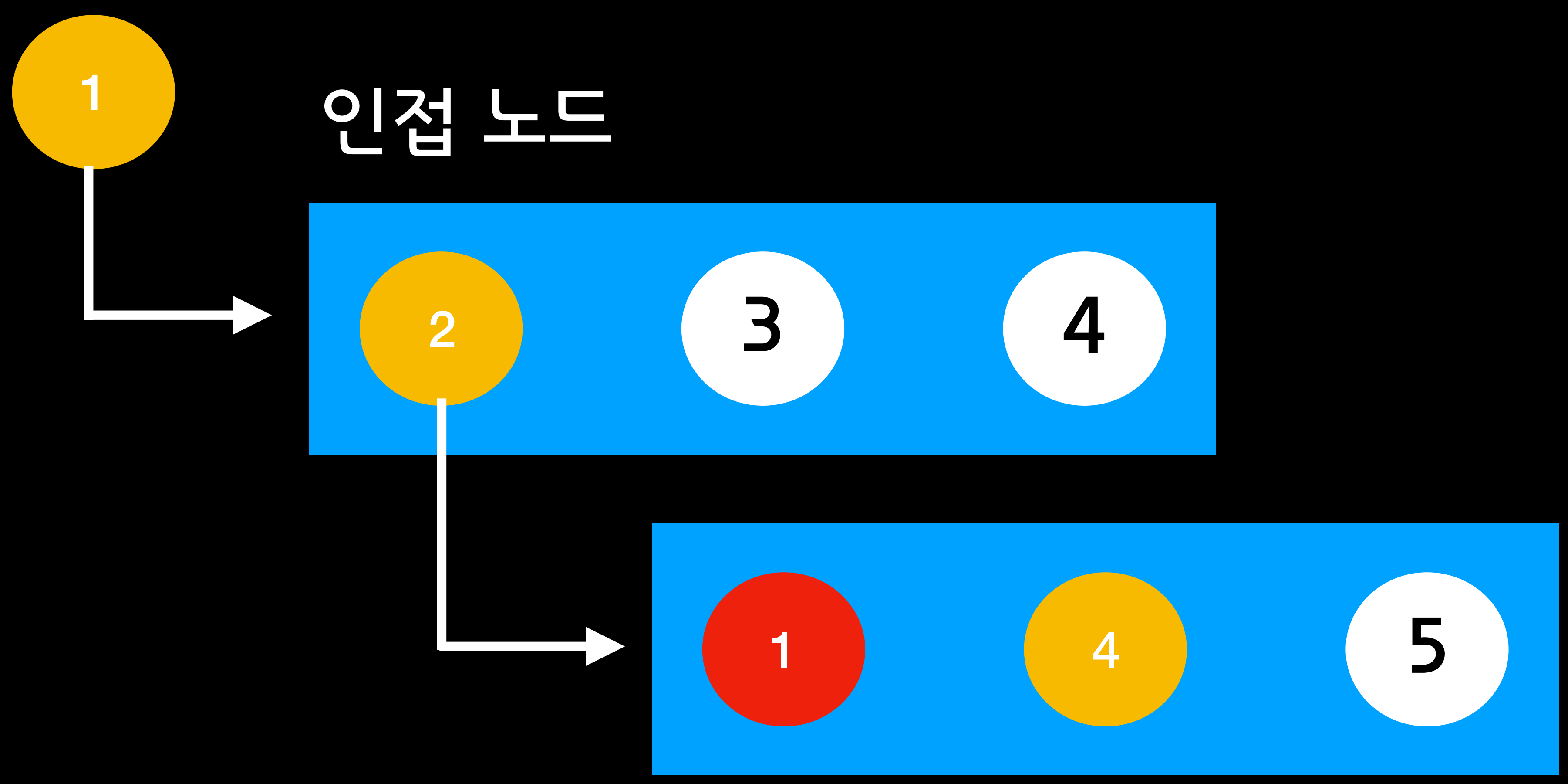
Examples

1번 노드를 시작으로 하여 최종 5번 노드까지의 재귀 탐색 과정을 한번 살펴보겠습니다.



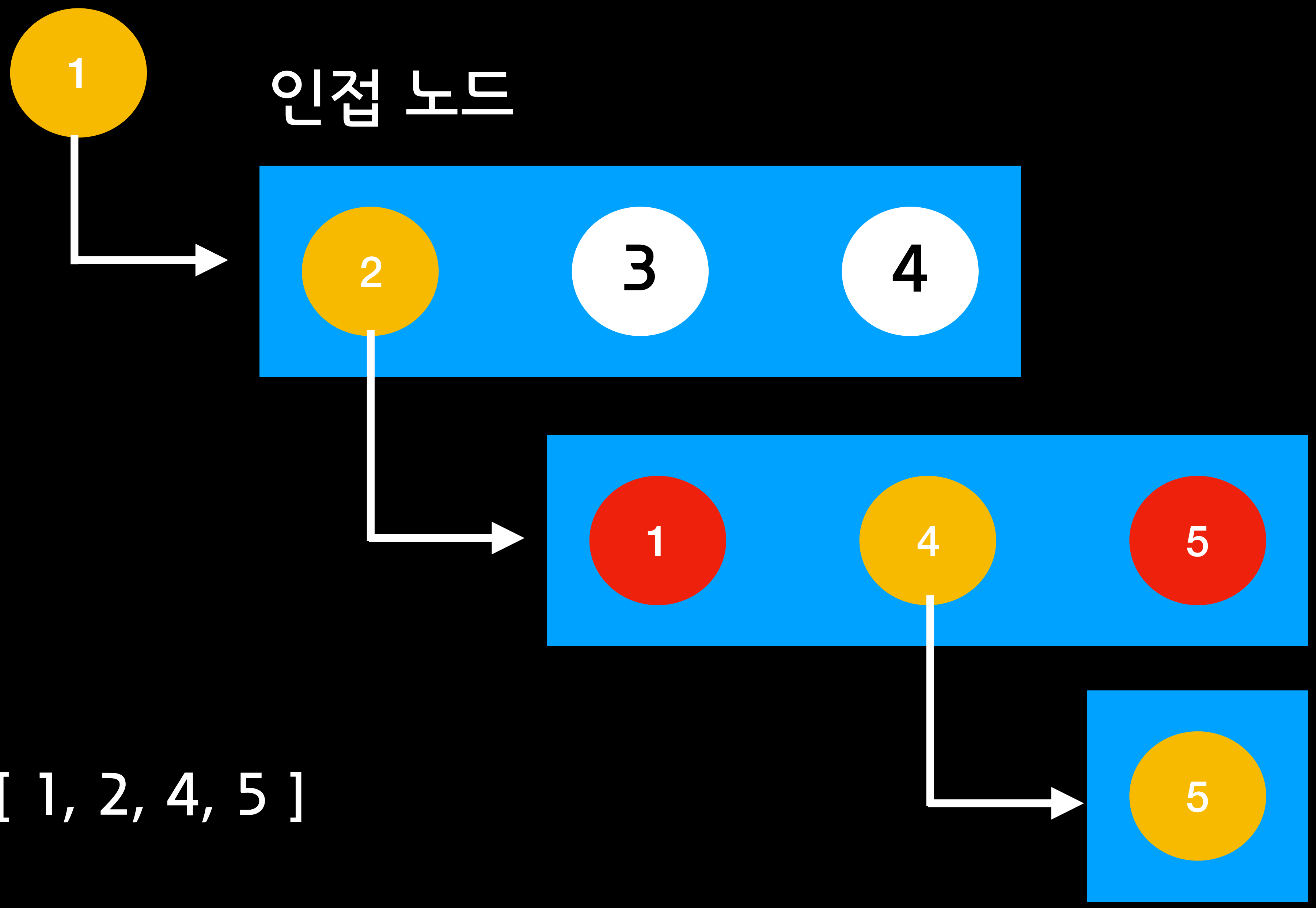
Examples

1번 노드를 시작으로 하여 최종 5번 노드까지의 재귀 탐색 과정을 한번 살펴보겠습니다.



Examples

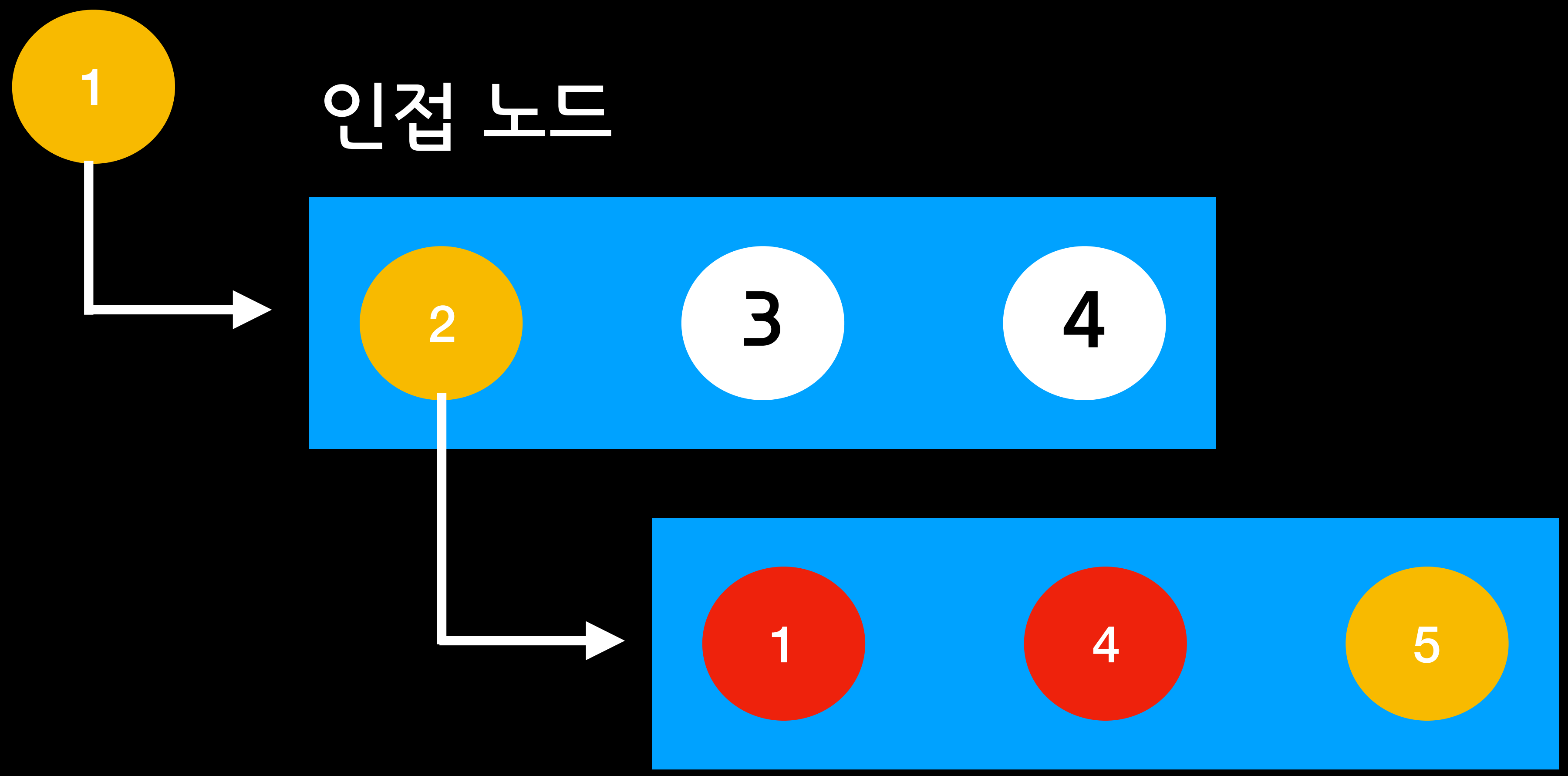
1번 노드를 시작으로 하여 최종 5번 노드까지의 재귀 탐색 과정을 한번 살펴보겠습니다.



[1, 2, 4, 5]

Examples

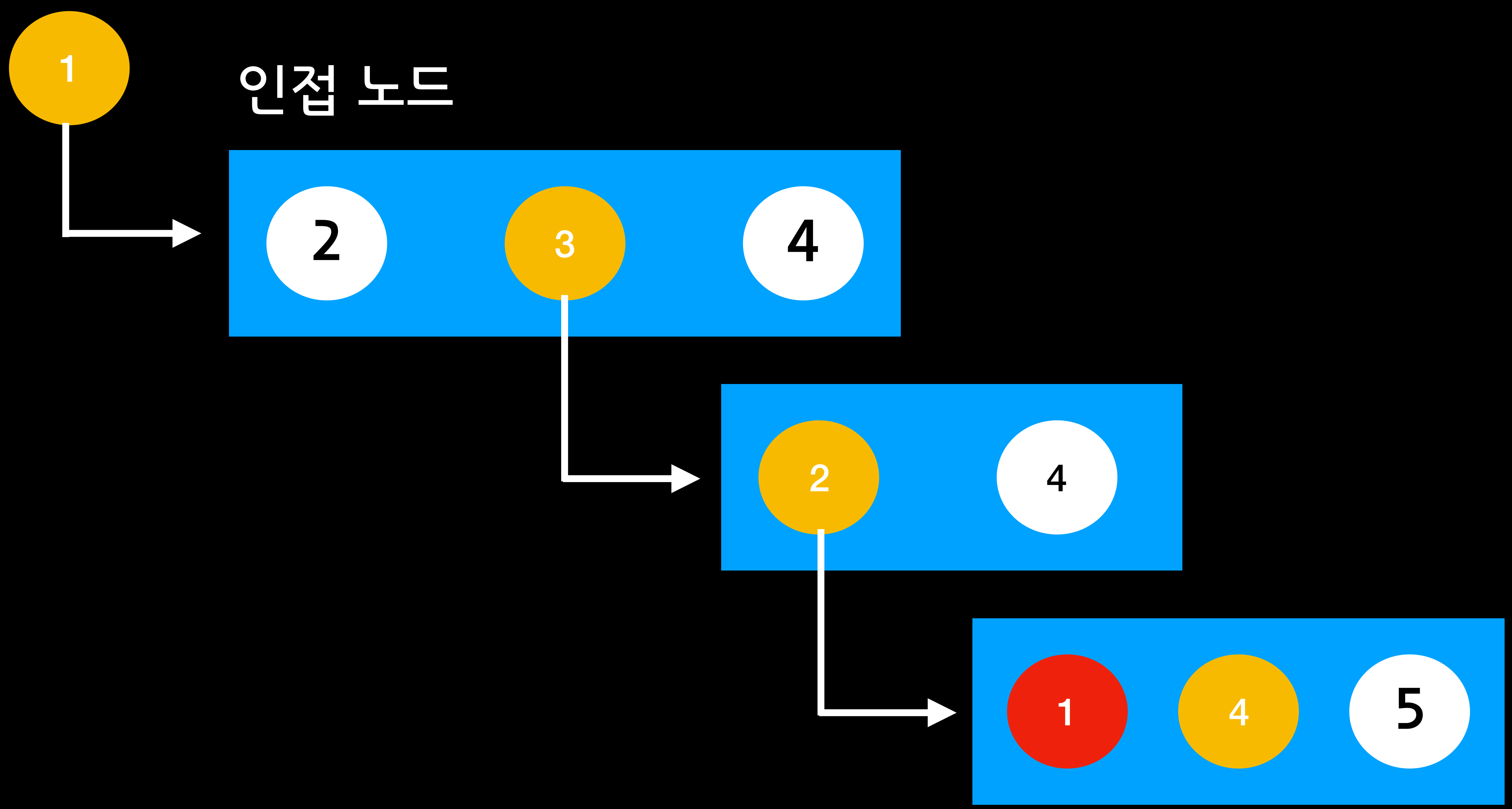
1번 노드를 시작으로 하여 최종 5번 노드까지의 재귀 탐색 과정을 한번 살펴보겠습니다.



[1, 2, 5]

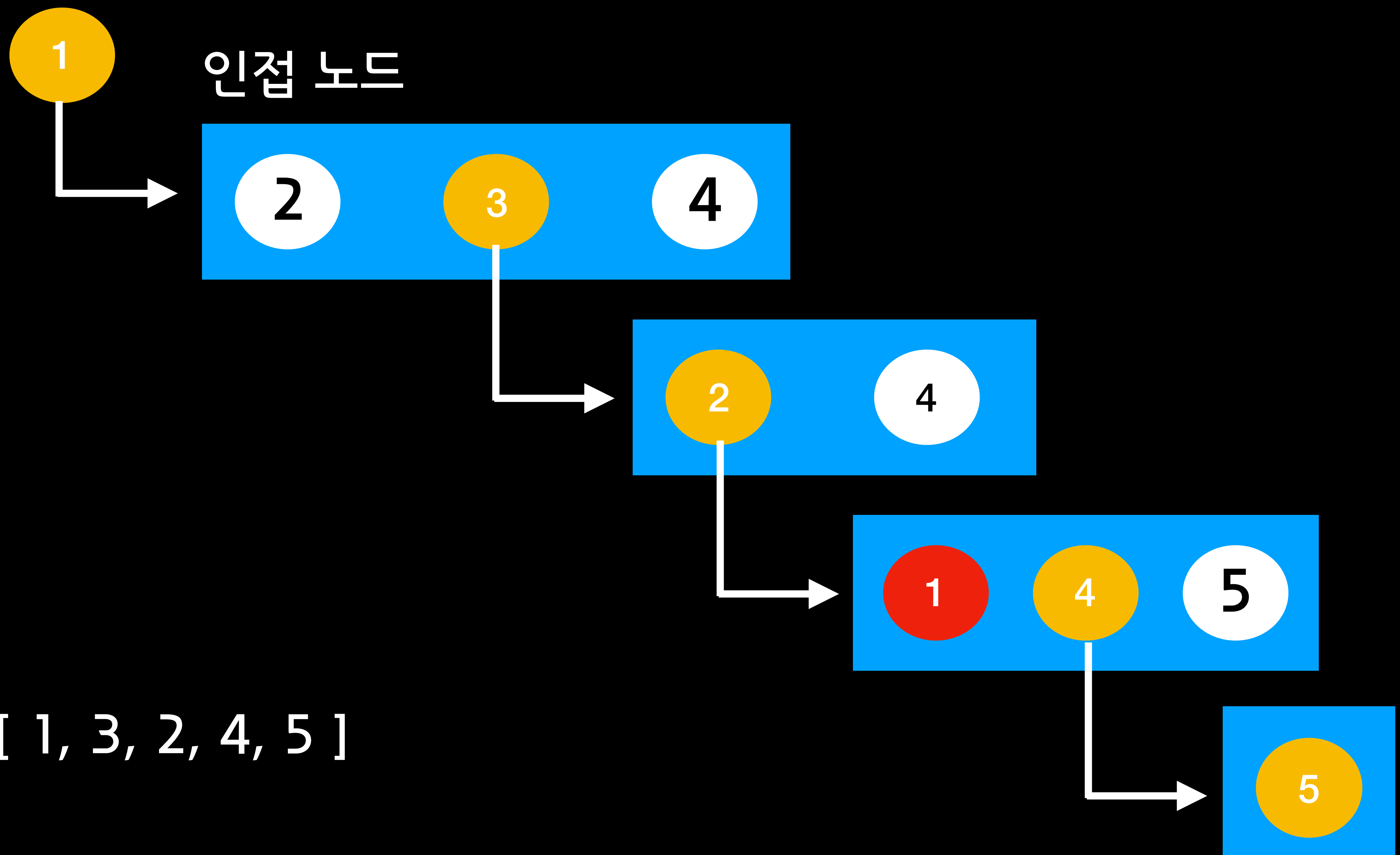
Examples

1번 노드를 시작으로 하여 최종 5번 노드까지의 재귀 탐색 과정을 한번 살펴보겠습니다.



Examples

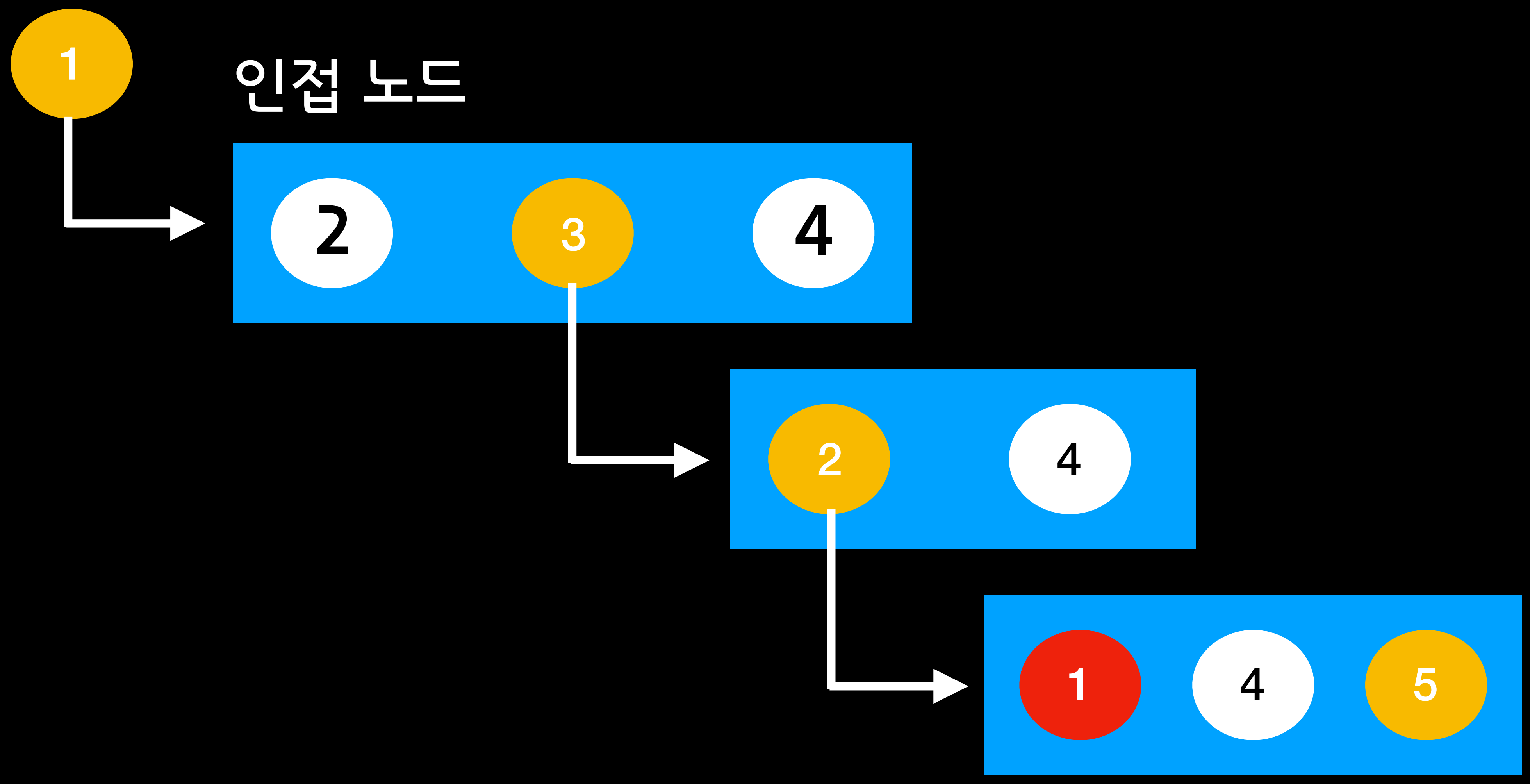
1번 노드를 시작으로 하여 최종 5번 노드까지의 재귀 탐색 과정을 한번 살펴보겠습니다.



[1, 3, 2, 4, 5]

Examples

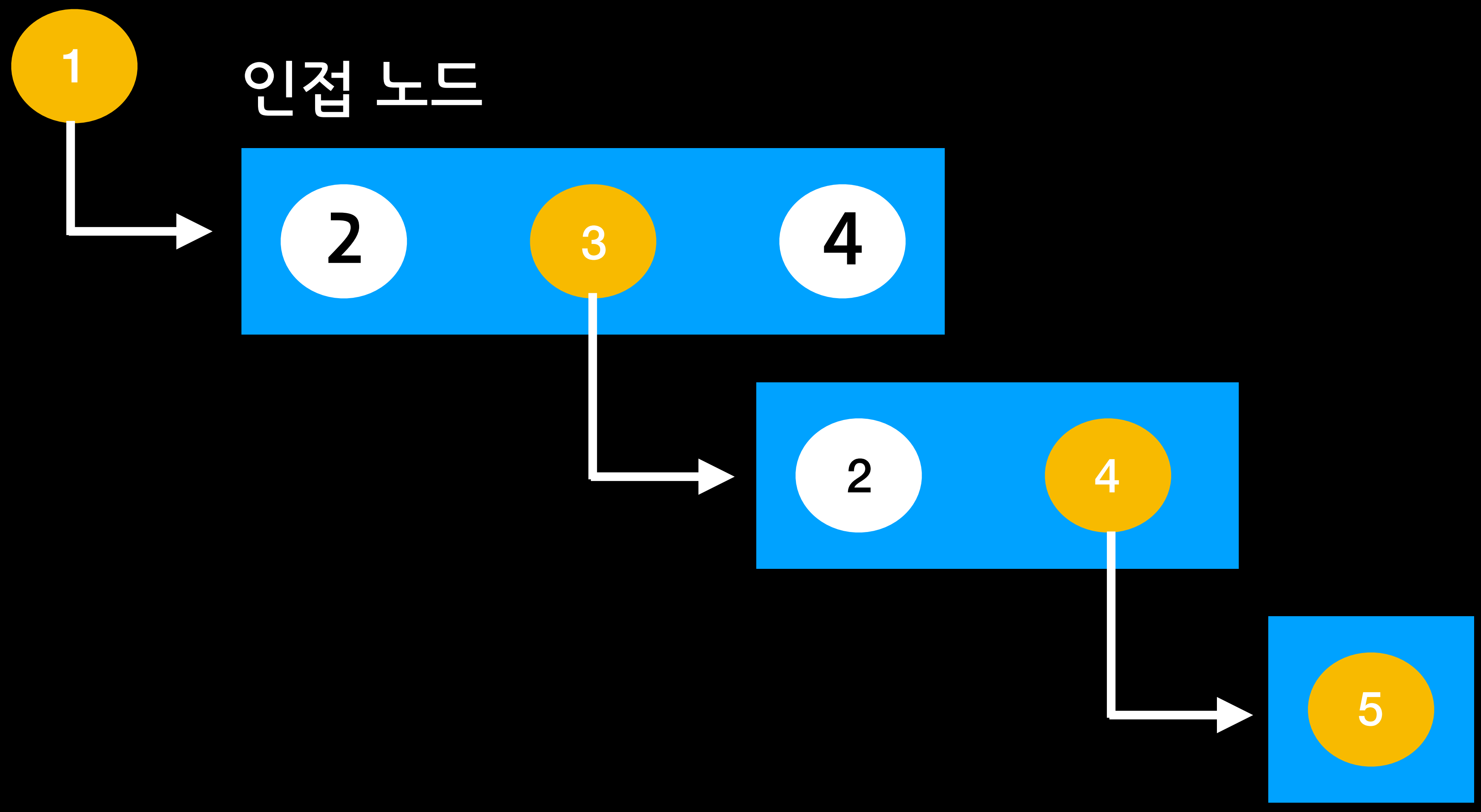
1번 노드를 시작으로 하여 최종 5번 노드까지의 재귀 탐색 과정을 한번 살펴보겠습니다.



[1, 3, 2, 5]

Examples

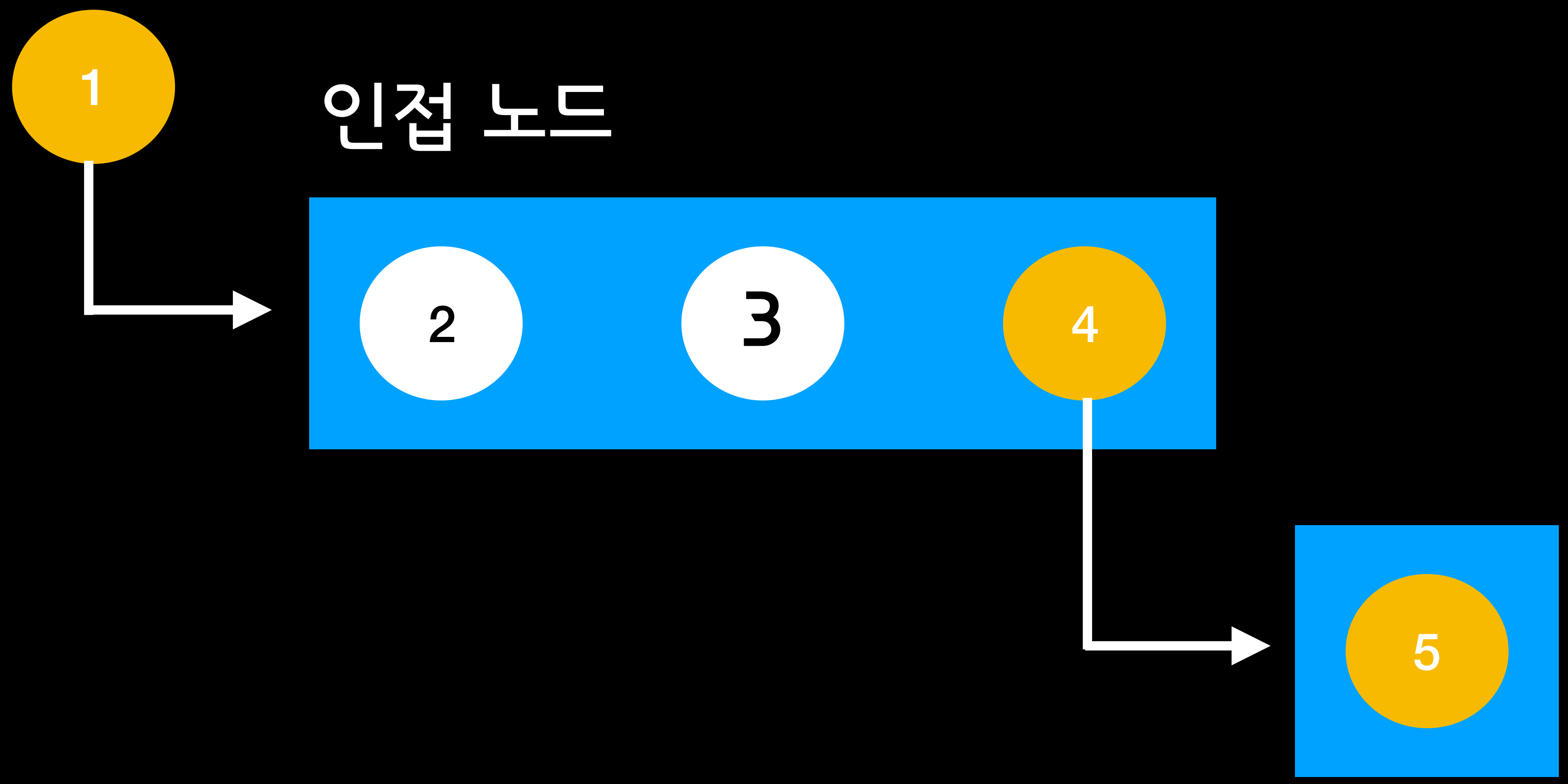
1번 노드를 시작으로 하여 최종 5번 노드까지의 재귀 탐색 과정을 한번 살펴보겠습니다.



[1, 3, 4, 5]

Examples

1번 노드를 시작으로 하여 최종 5번 노드까지의 재귀 탐색 과정을 한번 살펴보겠습니다.



[1, 4, 5]

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(AdjacencyListGraph) 객체

그래프 기본 메소드

- depthFirstSearch : DFS(깊이 우선 탐색)를 실행하는 함수

Implementation

```
public struct Vertex<T>: Equatable where T: Hashable {
    public var data: T
    public let index: Int
}

extension Vertex: CustomStringConvertible {
    public var description: String {
        return "\(index): \(data)"
    }
}

extension Vertex: Hashable {
    public func hash(into hasher: inout Hasher) {
        hasher.combine(data)
        hasher.combine(index)
    }
}

public func ==<T>(lhs: Vertex<T>, rhs: Vertex<T>) -> Bool {
    guard lhs.index == rhs.index else {
        return false
    }

    guard lhs.data == rhs.data else {
        return false
    }

    return true
}
```

Implementation

```
public struct Edge<T>: Equatable where T: Hashable {
    public let from: Vertex<T>
    public let to: Vertex<T>

    public let weight: Double?
}

extension Edge: CustomStringConvertible {
    public var description: String {
        guard let unwrappedWeight = weight else {
          return "\(from.description) -> \(to.description)"
        }

        return "\(from.description) -(\(unwrappedWeight))-> \(to.description)"
    }
}

extension Edge: Hashable {
    public func hash(into hasher: inout Hasher) {
        hasher.combine(from.description)
        hasher.combine(to.description)
        hasher.combine(weight)
    }
}

public func == <T>(lhs: Edge<T>, rhs: Edge<T>) -> Bool {
    guard lhs.from == rhs.from else {
        return false
    }

    guard lhs.to == rhs.to else {
        return false
    }

    guard lhs.weight == rhs.weight else {
        return false
    }

    return true
}
```

Implementation

```swift
open class AbstractGraph<T>: CustomStringConvertible where T: Hashable {
    public required init() {}

    public required init(fromGraph graph: AbstractGraph<T>) {
        for edge in graph.edges {
            let from = createVertex(edge.from.data)
            let to = createVertex(edge.to.data)

            addDirectedEdge(from, to: to, withWeight: edge.weight)
        }
    }

    open func createVertex(_ data: T) -> Vertex<T> {
        fatalError("abstract function called")
    }

    open func addDirectedEdge(_ from: Vertex<T>, to: Vertex<T>, withWeight weight: Double?) {
        fatalError("abstract function called")
    }

    open func addUndirectedEdge(_ vertices: (Vertex<T>, Vertex<T>), withWeight weight: Double?) {
        fatalError("abstract function called")
    }

    open func weightFrom(_ sourceVertex: Vertex<T>, to destinationVertex: Vertex<T>) -> Double? {
        fatalError("abstract function called")
    }

    open func edgesFrom(_ sourceVertex: Vertex<T>) -> [Edge<T>] {
        fatalError("abstract function called")
    }

    open var description: String {
        fatalError("abstract property accessed")
    }

    open var vertices: [Vertex<T>] {
        fatalError("abstract property accessed")
    }

    open var edges: [Edge<T>] {
        fatalError("abstract property accessed")
    }
}
```

Implementation

```
private class EdgeList<T> where T: Hashable {
    var vertex: Vertex<T>
    var edges: [Edge<T>]?

    init(vertex: Vertex<T>) {
        self.vertex = vertex
    }

    func addEdge(_ edge: Edge<T>) {
        edges?.append(edge)
    }
}
```

Implementation

```
open class AdjacencyListGraph<T>: AbstractGraph<T> where T: Hashable {
    fileprivate var adjacencyList: [EdgeList<T>] = []

    public required init() {
        super.init()
    }

    public required init(fromGraph graph: AbstractGraph<T>) {
        super.init(fromGraph: graph)
    }

    open override var vertices: [Vertex<T>] {
        var vertices = [Vertex<T>]()
        for edgeList in adjacencyList {
            vertices.append(edgeList.vertex)
        }
        return vertices
    }

    ....
```

Implementation

```
....

open override var edges: [Edge<T>] {
    var allEdges = Set<Edge<T>>()
    for edgeList in adjacencyList {
      guard let edges = edgeList.edges else {
        continue
      }

      for edge in edges {
        allEdges.insert(edge)
      }
    }
    return Array(allEdges)
}

open override func createVertex(_ data: T) -> Vertex<T> {
    // check if the vertex already exists
    let matchingVertices = vertices.filter { vertex in
        return vertex.data == data
    }

    if matchingVertices.count > 0 {
        return matchingVertices.last!
    }

    // if the vertex doesn't exist, create a new one
    let vertex = Vertex(data: data, index: adjacencyList.count)
    adjacencyList.append(EdgeList(vertex: vertex))
    return vertex
}
….
```

Implementation

```
....

open override func addDirectedEdge(_ from: Vertex<T>, to: Vertex<T>, withWeight weight: Double?) {
    let edge = Edge(from: from, to: to, weight: weight)
    let edgeList = adjacencyList[from.index]
    if edgeList.edges != nil {
        edgeList.addEdge(edge)
    } else {
        edgeList.edges = [edge]
    }
}

open override func addUndirectedEdge(_ vertices: (Vertex<T>, Vertex<T>), withWeight weight:
Double?) {
    addDirectedEdge(vertices.0, to: vertices.1, withWeight: weight)
    addDirectedEdge(vertices.1, to: vertices.0, withWeight: weight)
}

open override func weightFrom(_ sourceVertex: Vertex<T>, to destinationVertex: Vertex<T>) ->
Double? {
    guard let edges = adjacencyList[sourceVertex.index].edges else {
        return nil
    }

    for edge: Edge<T> in edges {
      if edge.to == destinationVertex {
        return edge.weight
      }
    }

    return nil
}
….
```

Implementation

```
open override func edgesFrom(_ sourceVertex: Vertex<T>) -> [Edge<T>] {
        return adjacencyList[sourceVertex.index].edges ?? []
    }

    open override var description: String {
        var rows = [String]()
        for edgeList in adjacencyList {

          guard let edges = edgeList.edges else {
            continue
          }

          var row = [String]()
          for edge in edges {
            var value = "\(edge.to.data)"
            if edge.weight != nil {
              value = "(\(value): \(edge.weight!))"
            }
            row.append(value)
          }

          rows.append("\(edgeList.vertex.data) -> [\(row.joined(separator: ", "))]")
        }

        return rows.joined(separator: "\n")
    }
}
```

Implementation

```
func depthFirstSearch(source: NodeGraph) -> [String] {
    var nodesExplored = [source.label]
    source.visited = true

    for edge in source.neighbors {
        if !edge.neighbor.visited {
           nodesExplored += depthFirstSearch(source: edge.neighbor)
        }
    }

    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)

let nodesExplored = depthFirstSearch(source: nodeA)
print(nodesExplored)
// ["A", "B", "D", "E", "H", "F", "G", "C"]
```

Implementation

```swift
func depthFirstSearch(start: Int, end lastNode: Int, edges: [(Int, Int)]) -> [Any] {
    var edgeInfo = [Int: Array<Int>]()

    for edge in edges {
        if var array = edgeInfo[edge.0] {
            array.append(edge.1)
            edgeInfo[edge.0] = array
        } else { edgeInfo[edge.0] = [edge.1] }
    }

    var result = 0
    var paths:[[Any]] = [[]]

    func dfs(node: Int, visited: [Int]) {
        guard node != lastNode else {
            if result < lastNode { paths.append(visited) }
            result += 1
            return
        }

        guard let neighbors = edgeInfo[node] else { return }

        for edge in neighbors {
            guard visited.contains(edge) == false else { continue }

            dfs(node: edge, visited: visited + [edge])
        }
    }

    dfs(node: start, visited: [1])

    return paths.filter { $0.count != 0 }
}
```

Implementation

```
print(depthFirstSearch(start: 1,
                         end: 5,
                       edges: [(1, 2), (1, 3), (1, 4), (2, 1),
                               (2, 4), (2, 5), (3, 2), (3, 4), (4, 5)]))

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

References

[1] Swift로 그래프 탐색 알고리즘을 실전 문제에 적용해보기 - DFS 편 : https://wlaxhrl.tistory.com/88?category=842165

[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] 깊이 우선 탐색 : https://ko.wikipedia.org/wiki/깊이_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] Swift) Graph의 DFS를 이용한 경로 찾기 : https://velog.io/@dusdl14/Swift-Graph의-DFS를-이용한-경로-찾기

[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!