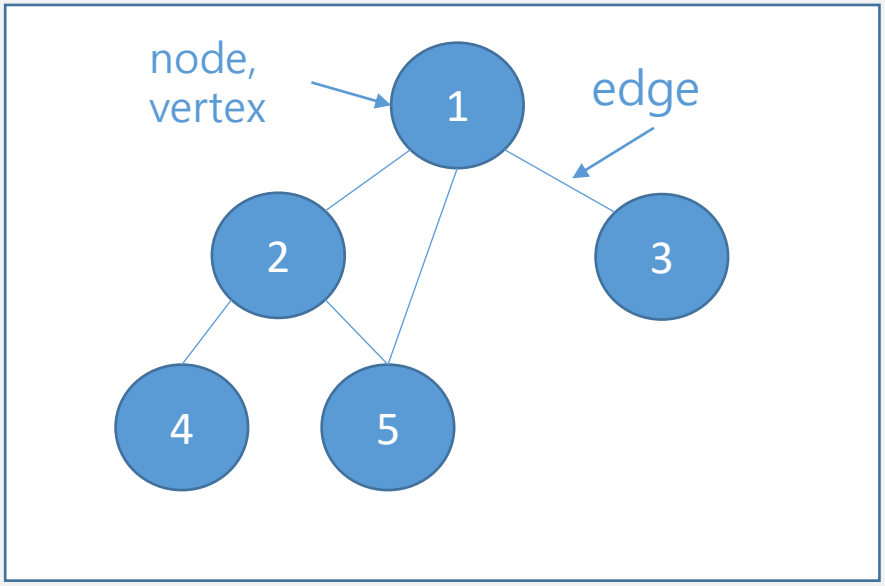
Graph

BFS & DFS

Shrtest Path

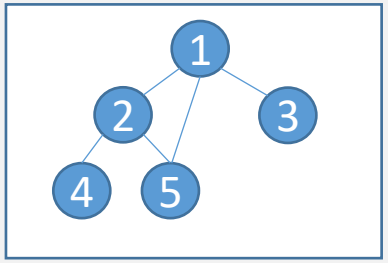
그래프

- 그래프(Graph)
  - 연결되어 있는 객체 간의 관계를 표현하는 비선형 자료구조
  - 트리는 사이클이 없는 그래프



그래프(Graph)

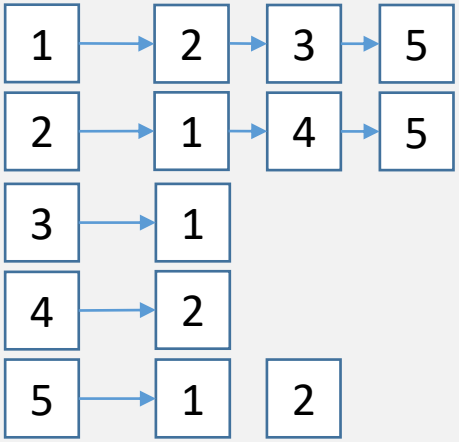
■ 그래프 표현방법



인접행렬(adjacency matrix)

$$\begin{bmatrix} 0 & 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \end{bmatrix}$$

인접리스트(adjacency List)

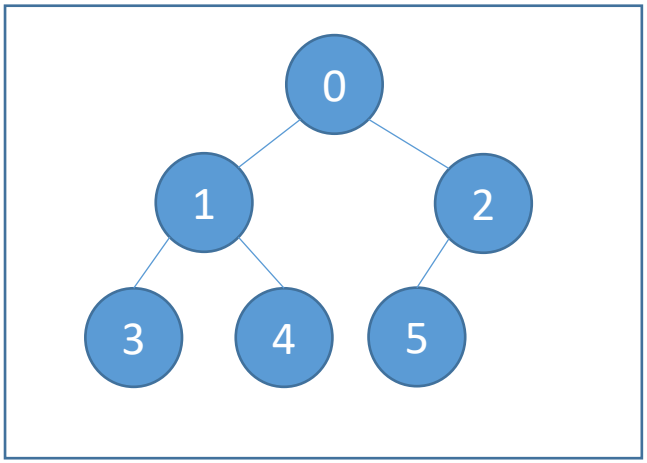


사전(dictionary)

```
g = {
    1: [2, 3, 5],
    2: [1, 4, 5],
    3: [1],
    4: [2],
    5: [1, 2]
}
```

너비 우선 탐색 Breadth First Search, BFS

queue 를 이용



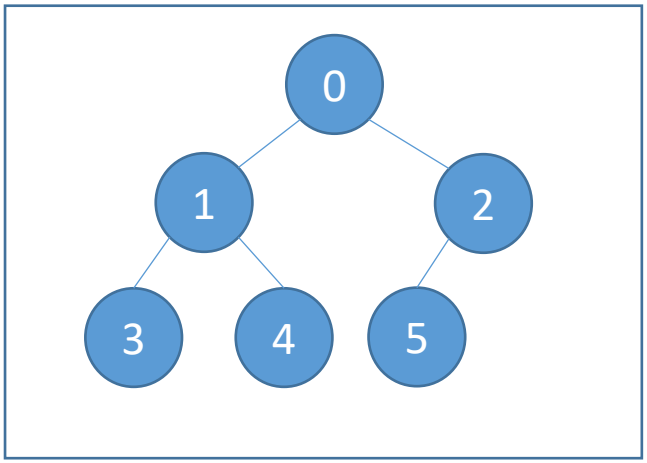
탐색순서 : 0, 1, 2, 3, 4, 5

or 0, 2, 1, 5, 4, 3

```
public void BFS(int v)    {
    bool[] visited = new bool[V];
    Queue<int> queue = new Queue<int>();

    visited[v] = true;
    queue.Enqueue(v);

    while (queue.Count != 0)        {
        v = queue.Dequeue();
        Console.Write(v + " ");

        foreach (int i in adj[v])            {
            if (!visited[i])                {
                visited[i] = true;
                queue.Enqueue(i);
            }
        }
    }
}
```

깊이 우선 탐색 Depth First Search, DFS

stack을 이용



탐색순서 : 0, 1, 3, 4, 2, 5

or 0, 2, 5, 1, 4, 3

```
public void DFS(int v)    {
    bool[] visited = new bool[V];
    Stack<int> stack = new Stack<int>();

    visited[v] = true;
    stack.Push(v);

    while (stack.Count != 0)        {
        v = stack.Pop();
        Console.Write(v + " ");

        foreach (int i in adj[v])            {
            if (!visited[i])                {
                visited[i] = true;
                stack.Push(i);
            }
        }
    }
}
```

DFS 재귀 함수

```
public void DFS_Recursion(int v)    {
    bool[] visited = new bool[V];

    DFSUtil(v, visited);
}
// 자신을 방문한것으로 하고 인접한 정점에 대해 재귀
private void DFSUtil(int v, bool[] visited)    {
    visited[v] = true;
    Console.Write(v + " ");

    foreach (int i in adj[v])
    {
        if (!visited[i])
        {
            DFSUtil(i, visited);
        }
    }
}
```

class Graph

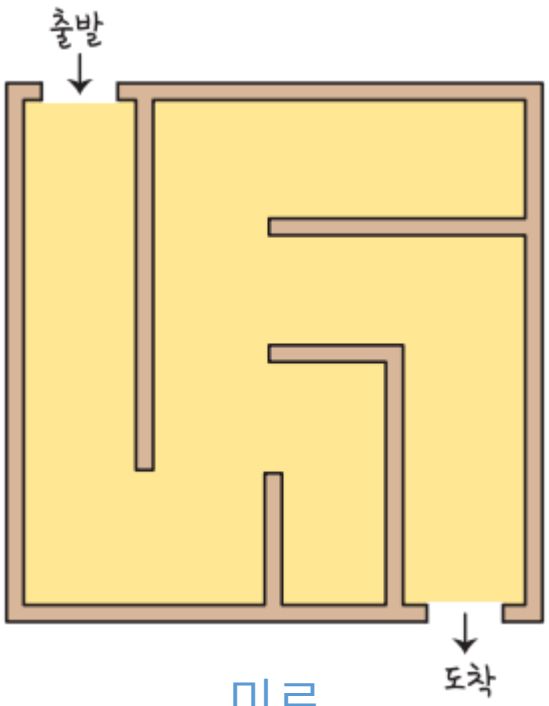
```
class Graph{
    private int V; // 그래프의 정점 개수
    private List<int>[] adj; // 인접 리스트
    public Graph(int v)    {
        V = v;
        adj = new List<int>[V];
        for (int i = 0; i < V; i++)        {
            adj[i] = new List<int>();
        }
    }
    public void AddEdge(int v, int w)    {
        adj[v].Add(w); // 정점 v에서 w로 가는 간선 추가
    }
```

Main

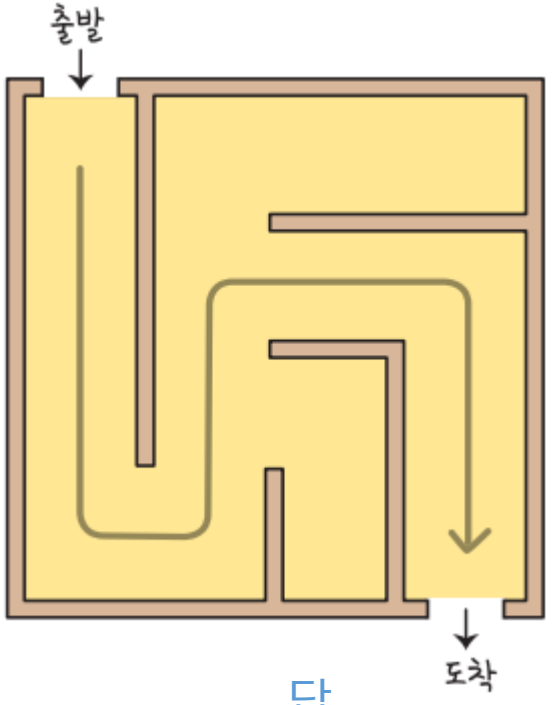
```
static void Main(string[] args)    {
    Graph graph = new Graph(6); //Vertex
    graph.AddEdge(0, 1);
    graph.AddEdge(0, 2);
    graph.AddEdge(1, 3);
    graph.AddEdge(1, 4);
    graph.AddEdge(2, 5);

    Console.WriteLine("BFS 탐색 결과:");
    graph.BFS(0);

    Console.WriteLine("₩nDFS 탐색 결과:");
    graph.DFS(0);
}
```

미로 찾기 알고리즘

다음 그림과 같이 미로와 출발점, 도착점이 주어졌을 때 출발점에서 도착점까지 가기 위한 최단 경로를 찾는 알고리즘을 만드시오.



미로

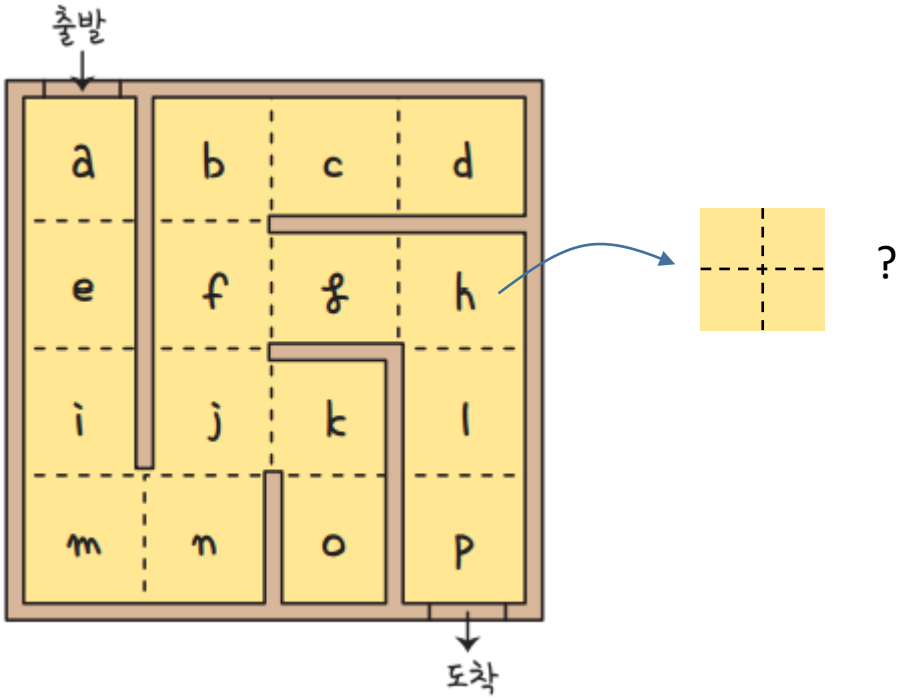


답

미로 찾기 알고리즘

문제: 출발점 a에서 벽으로 막히지 않은 위치로 차례로 이동하여 도착점 p에 이르는 가장 짧은 경로를 구하고, 그 과정에서 지나간 위치의 이름 출력.

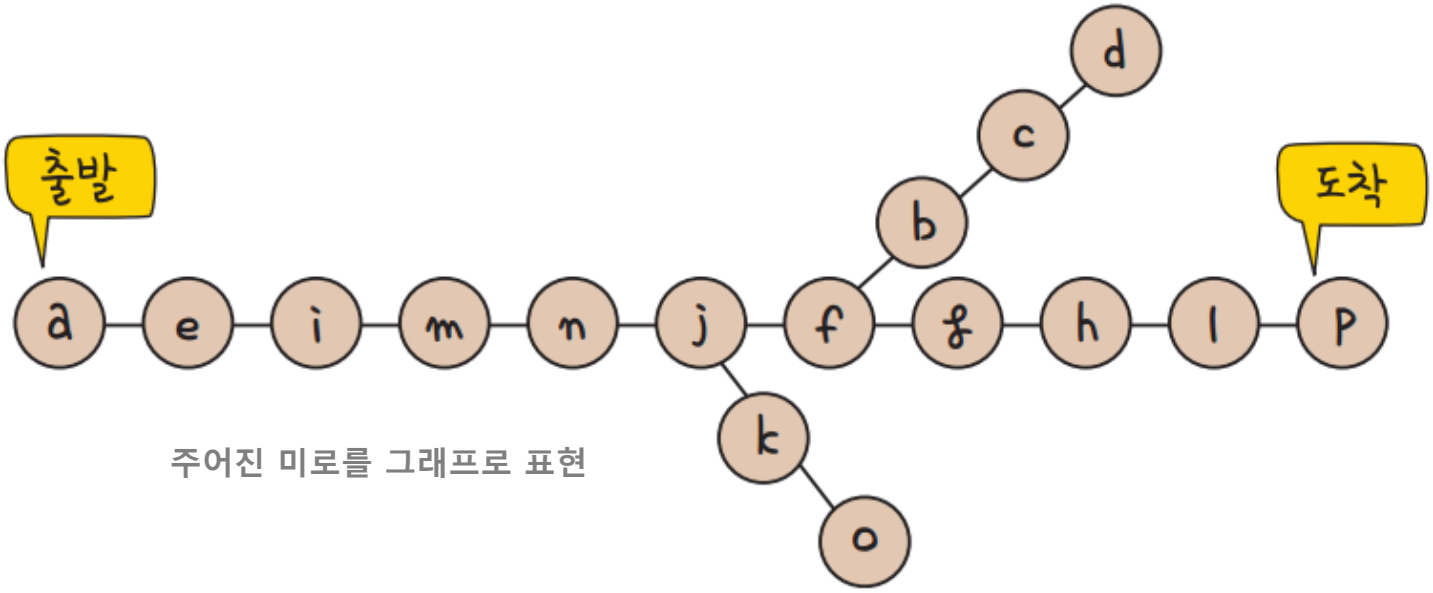
정답: aeimnjfghlp



- 그래프를 이용해 미로 찾기 문제를 단계별로 모델링
- 일단 미로를 풀려면 미로 안의 공간을 **정형화**해야 함

문제 분석과 모델링

- 이 문제는 그래프 탐색 문제와 같음
- 위치 열여섯 개를 각각 꼭짓점으로 만들고, 각 위치에서 벽으로 막히지 않아 이동할 수 있는 이웃한 위치를 모두 선으로 연결



주어진 미로를 그래프로 표현

미로 찾기 알고리즘

- 그래프를 딕셔너리(key : 정점, value : 인접한 정점) 로 변환

- 현실 세계의 문제를 컴퓨터로 풀려면 문제를 분석하여 효과적인 모델(모형)을 만드는 것이 가장 중요
- 먼저 문제를 잘 모델링하고, 그 모델에 여러 가지 알고리즘을 적용하여 문제를 푼 다음 그 결과를 다시 실제 세계에 적용하는 것
- 이는 실생활의 문제를 컴퓨터를 사용해서 푸는 일반적인 과정

알고리즘

- **너비우선 탐색 문제와 유사**
- 이동경로들과 방문한 곳을 저장할 큐와 집합 만듦
- 처리할 이동경로들이 있으면
  - 제일 앞의 경로를 뽑아 마지막 문자가 도착점이면 경로 반환 후 종료
  - 대상 위치에 연결된 위치들 중에 방문하지 않은곳들에 대해('j': ['f', 'k', 'ㅎ'])
    - 이동 경로에 새 위치로 추가하여 큐와 집합에 저장
    - qu: ['a> e> i> m> n> j> f', 'a> e> i> m> n> j> k']
- 나갈 수 없는 미로이므로 종료