13주차 예비보고서

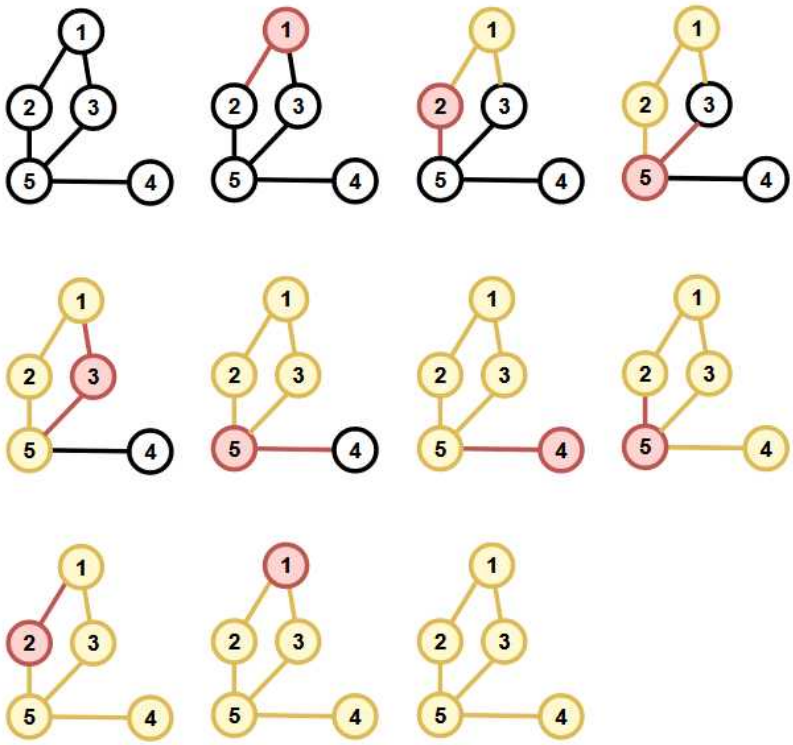
전공: 신문방송학과 학년: 3학년 학번: 20191150 이름: 전현길

1. DFS와 BFS의 시간 복잡도를 계산하고 그 과정을 설명한다.

DFS(Depth First Search; 깊이 우선 탐색)와 **BFS(Breadth First Search; 너비 우선 탐색)**는 그래프 탐색의 방법론으로, 시작 정점에서 방문할 수 있는 모든 정점을 탐색하는 알고리즘이다. 두 알고리즘은 스택, 큐라는 대표적인 자료구조를 활용해서 구현된다.

DFS 알고리즘의 작동 방식은 다음과 같다.

1. 임의의 정점에 방문한 뒤, 방문 여부를 표시하고 스택에 저장한다.
2. 현재 정점에 인접한 정점 중, 미방문한 정점이 있다면 방문한 뒤, 해당 정점에 대해 다시 1, 2를 반복한다.
3. 미방문한 정점이 없다면 스택의 값을 버리고, 1, 2를 반복한다.
4. 모든 정점에 대해 미방문한 정점이 없다면 알고리즘을 종료한다.



위의 그림은 인접한 노드 중 더 숫자가 작은 노드를 먼저 방문하는 DFS 알고리즘의 작동 과정을 표현한 그림이다. 그림에서 살펴볼 수 있듯이, DFS 알고리즘은 모든 정점을 한 번씩 탐색하는 과정에서 자신의 정점과 연결된 모든 간선을 한 번씩 순회하게 된다. 따라서 **시간 복잡도는 인접 행렬로 표현될 경우 O(V^2)이 되며, 인접 리스트로 표현될 경우 O(V + E)**로 표현된다.

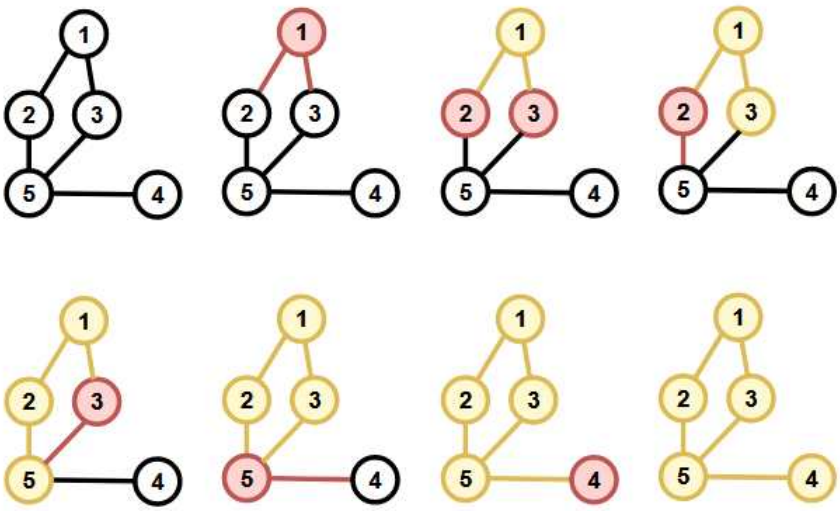
이 때 간선의 값이 V^2에 비례하므로 O(V^2)이라고 볼 수도 있지만, 간선이 적은 sparse graph일 경우 자료구조별 시간 복잡도 차이가 크게 나타나게 된다.

BFS 알고리즘의 작동 방식은 다음과 같다.

1. 임의의 정점에 방문한 뒤, 자기 자신을 큐에서 삭제한다.

2. 현재 정점에서 인접한 모든 미방문한 정점을 큐에 저장한 뒤, 저장한 정점에 대해 방문 여부를 표시한다. 다음 정점으로 이동해 1, 2를 반복한다.

3. 큐가 완전히 비면 알고리즘을 종료한다.



위의 그림은 인접한 노드 중 더 숫자가 작은 노드를 먼저 큐에 추가하는 BFS 알고리즘의 작동 과정을 표현한 그림이다. 정점은 큐의 FIFO 특성에 따라 큐에 추가된 순서대로 그래프를 순회하게 된다.

모든 정점을 한 번씩 탐색하는 과정에서 각 정점과 연결된 모든 간선을 한 번씩 확인하게 되므로, **DFS 알고리즘과 시간 복잡도 상 차이는 없다.** 단, BFS의 특성상 가짓수가 급격하게 증가하는 그래프의 경우 공간 복잡도가 증가하면서 메모리에 부담이 가해질 수 있다. 반대로 DFS의 경우 스택이 매우 깊이 쌓이는 경우 비슷한 문제가 발생한다.

2. 자신이 구현한 자료구조 상에서 DFS와 BFS 방법으로 실제 경로를 어떻게 찾는지 설명한다. 특히 DFS 알고리즘을 iterative한 방법으로 구현하기 위한 방법을 생각해보고 제시한다.

1) iterative DFS 알고리즘의 경로 계산

a. 스택 자료형에 현재 정점의 좌표를 저장하고, 방문 여부를 표시한다.

b. 현재 이동 가능한 정점의 좌표를 스택에 저장한 뒤 **즉시 종료(다음 loop로 이동)**한다. 이동 가능한 정점이 없다면, 현재 노드를 스택에서 삭제한다.

c. 도착점에 도착할 때까지 새로운 정점에 대해 a, b를 반복한다.

d. 도착점에 도착했다면, 스택이 빌 때까지 최단 경로를 구별할 수 있게 방문 체크하면서 노드를 삭제한다.

반복문을 활용한 DFS 알고리즘의 의사 코드는 다음과 같다. 재귀적 DFS 알고리즘의 경우 이보다 훨씬 간단해지는데, 미방문한 노드가 있을 경우 방문 처리하고 해당 노드에 대해 dfs()를 호출하는 코드를 작성하면 된다.

iterative DFS

```
stack.push(first vertex)
/* mark first vertex */
while (stack is not empty)
  if (arrived to destination)
    while (stack is not empty)
      current_location = stack.top()
      /* mark shortest path */
      stack.pop()
    break;
  if (univisited vertex exists)
    stack.push(unvisited vertex vertex)
    /* mark visited vertex */
  else
    stack.pop()
```

2) BFS 알고리즘의 경로 계산

a. 큐에 현재 정점의 좌표를 저장하고, 방문 여부를 표시한다.

b. 현재 정점에서 이동 가능한 정점의 좌표를 큐에 모두 저장한 뒤 현재 정점을 큐에서 삭제한다.

c. 큐가 빌 때까지 a, b를 반복한다.

BFS

```
queue.push(first vertex)
/* mark first vertex */
while (queue is not empty)
  queue.pop()
  if (univisited vertex exists)
    queue.push(unvisited vertex vertex)
    /* mark visited vertex */
```

BFS 알고리즘은 iterative DFS를 구현했던 것과 거의 동일하게 구현한다. 차이점이 있다면, 큐를 반복문이 시작할 때 즉시 삭제한다. 또 현재 이동 가능한 정점의 좌표를 스택에 저장한 뒤 즉시 종료하는 대신, 현재 이동 가능한 정점의 좌표를 모두 저장한 뒤 종료한다. 단 iterative DFS와 달리 최단 경로를 찾도록 구현하는 것이 어려워질 수 있다. 이는 각 정점마다 경로를 저장하도록 해서 해결할 수 있지만, 메모리를 많이 소모할 수 있다.