알고리즘 실습(7주차)

201000287 일어일문학과 유다훈

1. 과제 설명 및 해결 방법

1. Graph 구현

① Breadth First Search

- 입력 받은 Adjacent matrix를 이용하여 그래프를 넓이우선탐색을 한다.
- 어떠한 Vertex와 Edge로 연결된 Vertex를 차례차례 탐색해나가는 방법.

② Depth First Search

- 입력 받은 Adjacent matrix를 이용하여 그래프를 깊이우선탐색을 한다.
- 어떠한 Vertex와 Edge로 연결된 Vertex를 한 쪽으로만 계속 찾아가며 검색을 한다.
- 이 후 더이상 연결된 Vertex가 없을 때는 다른 Vertex와도 연결되어 있는 상위 Vertex까지 Backtrack하여 거슬러 올라간 후, 그 곳에서 다시 다른 Vertex를 검색하기 시작한다.

2. 주요 부분 코드 설명(알고리즘 부분)

1. Breadth First Search

```
void Breadth_First_Search(int **matrix, int n ) {

    int visited[n]; //방문한 vertex를 재방문하지 않기 위한 확인 방법. 배열선언.

    for (int i =0; i < 8; i++) {
        visited[i] = 0; //배열의 초기화. 0은 방문하지 않은 것, 1은 이미 방문 한 것.
                        //처음에는 모두 방문하지 않았음.
    }

    enQueue(0);
    visited[0] = 1; //첫 방문은 임의로 시작하나 이번 과제에서는 0번 노드부터 시작.

    while(queue->value != -1 || queue->next != NULL) {
                                            //큐가 비어있지 않을 동안 혹은 큐의 다음값이 널이 아닌 동안
        int i = deQueue();                  //큐에서 값 하나를 추출
        printf("%d\n", i);                  //출력
        for (int j = 0; j < n; j++){        //데이터의 갯수만큼 진행
            if (visited[j] == 0 && matrix[i][j] == 1) { //방문을 안했으면서 연결된 버텍스라면
                enQueue(j); //큐에 집어넣고
                visited[j] = 1; //큐에 집어넣은 것은 방문한 것
            }
    }

}
}
```

- 입력받은 adjacent matrix를 가지고 있는 2차원 배열과 데이터의 갯수를 인자로 받는 함수로 만든다.
- 각각의 노드의 방문 여부를 확인하기 위해 visited라는 이름의 방문 확인용 배열을 생성하고 초기화를 해준다.
    - 해당 배열의 값이 0이면 방문을 안한 것이고, 1이라면 방문을 한 것이다.
- 큐를 이용하여 방문한 노드와 인접 노드들을 방문한다.
- 0번노드부터 방문을 하기위해 큐에 집어 넣는다. 동시에 visited 배열의 0번자리에 1을 넣어준다.
- 큐가 비어있지 않거나 큐의 기본값의 다음 값이 널이 아닐 동안 반복을 한다.
    - 큐에서 값을 하나 빼고, 이 값은 방문이 끝난 값이므로 출력해준다.
    - 자료의 갯수만큼 반복문을 돌려준다.
    - 반복문의 변수값에 방문한 적이 없으면서, 큐에서 뽑아낸 값과 연결된 값이라면 큐에 집어넣고 방문했다는 표시로 visited배열에 1을 넣어준다.
- 이 행위를 계속하여 반복해준다.

2. Depth First Search

```
void DFS_Visit(int **matrix, int *visited, int i, int n) {

    visited[i] = 1;
    printf("%d\n", i);


        for(int j=0; j<n; j++) {
            if (visited[j] == 0 && matrix[i][j] == 1) {
                DFS_Visit(matrix, visited, j, n);
            }
    }

}
void Depth_First_Search(int **matrix, int n) {

    int visited[n]; //방문한 vertex를 재방문하지 않기 위한 확인 방법. 배열선언.

    for (int i =0; i < 8; i++) {
        visited[i] = 0; //배열의 초기화. 0은 방문하지 않은 것, 1은 이미 방문 한 것.
    }

        DFS_Visit(matrix, visited, 0, n);

}
```

- 입력 받은 adjacent matrix가 저장되어있는 2차원 배열과 방문해야 할 전체 데이터의 갯수를 인자로 넘겨받는 함수 Depth_First_Search를 선언한다.
- 방문했는지의 여부를 표시하기 위해 데이터 갯수 만큼의 크기를 가진 배열 visited를 선언하여 초기화를 해준다. 값이 0이라면 아직 방문하지 않은 것이고, 1이라면 방문한 것이다.
- 이후 데이터가 들어있는 2차원 배열과 방문여부를 확인하는 1차원 배열, 방문할 노드, 자료의 갯수를 인자로 받는 DFS_Visit 함수를 호출한다.
- DFS_Visit를 실행하면, 인자로 전달받는 방문할 노드의 방문여부를 1로 표시하여 방문한 것으로 확인한다.
- 이후 반복문을 이용하여 방문하지 않았으며, 인자로 전달받은 노드와 연결된 노드가 있다면 DFS_Visit를 재귀적으로 호출해준다.
    - 이렇게 된다면, 어떠한 값 i의 2차원 배열값에 많은 노드들이 연결되어 있더라도 가장 처음 만나는 연결된 노드값에 대하여 재귀적인 함수 호출을 하여 처음 만나는 노드를 따라 계속해서 찾아가는 깊이우선탐색을 할 수 있다.

3. 결과

- Breadth First Search
  - 인접한 노드들을 Adjacent matrix의 순서대로 방문한다.
- Depth First Search
  - 연결된 노드들 중에서 한 쪽으로 계속해서 찾아 검색하는 깊이우선탐색을 한다.
- 두 탐색 방법의 특성상 같은 입력이나 결과물은 달라진다.