

## 자료구조2 실습 레포트



과목명		자료구조2 실습
담당교수		홍 민 교수님
학과		컴퓨터소프트웨어공학과
학년		2학년
학번		20204059
이름		이예빈

# 목 차

---

## 1. 그래프의 깊이 우선 탐색 (DFS: Depth First Search)

### 1.1 문제 분석

#### 1.1.1 그래프의 인접 행렬 생성 과정

#### 1.1.2 깊이 우선 탐색 과정 그림

### 1.2 소스 코드

### 1.3 소스 코드 분석

### 1.4 실행창

### 1.5 느낀점

## 2. 그래프 너비 우선 탐색 (BFS: Breadth First Search)

### 2.1 문제 분석

#### 2.1.1 너비 우선 탐색 과정 그림

### 2.2 소스 코드

### 2.3 소스 코드 분석

### 2.4 실행창

### 2.5 느낀점

## 3. 느낀점

## 1. 인접행렬로 구현하는 지하철 노선 그래프



### 그래프

#### ■ 그래프 깊이 우선 탐색 프로그램

- 382 페이지에 있는 프로그램 10.3의 깊이 우선 탐색 프로그램을 참고하여 파일에 입력되어있는 정점과 간선의 정보를 이용하여 그래프를 구성하고 이 그래프를 깊이 우선 탐색을 통해 출력하는 코드를 작성하시오.
  - 레포트 제출시 그래프가 그려지는 과정과 함께 깊이를 탐색하는 순서를 그린 그림과 함께 제출 (PPT, 한글, WORD 로 그린 그림 제출)

C:\WINDOWS\system32\cmd.exe

- 그래프 깊이 우선 탐색 결과 -

< 0 1 4 3 5 2 7 >

계속하려면 아무 키나 누르십시오 . . .

data - 메모장

파일(F) 편집(E) 서식(O) 보

v 0	e 2 0
v 1	e 2 5
v 2	e 2 7
v 3	e 3 0
v 4	e 3 4
v 5	e 4 1
v 7	e 4 3
e 0 1	e 5 1
e 0 2	e 5 2
e 0 3	e 5 7
e 1 0	e 7 2
e 1 4	e 7 5
e 1 5	

#### 1.1 문제 분석

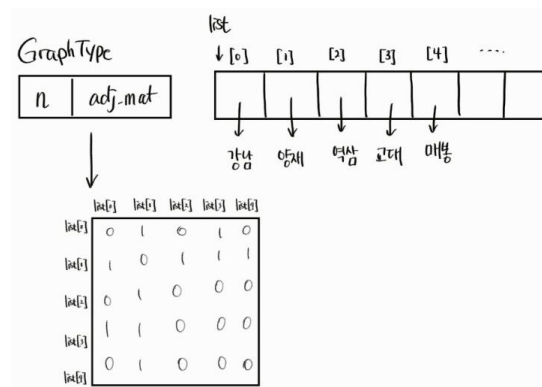
1번 문제는 파일로부터 정점과 간선의 정보들을 읽어 그래프를 생성하고, 이 그래프를 이용해 깊이 우선 탐색(Depth First Search)을 하여 방문한 정점들을 출력하는 프로그램이다. 그래프를 구현하기 위해서는 인접 행렬 또는 인접 리스트를 사용할 수 있다. 예시 결과창의 깊이 우선 탐색 결과와 동일한 결과를 얻기 위해서는 인접 행렬을 이용하여 그래프를 구현하는 것이 적합하다고 판단하였다. 따라서 '인접 행렬'을 이용하여 정점들의 관계를 나타내도록 프로그램을 구현하도록 하자. 행렬을 이용해 그래프를 구현할 때, 메모리의 낭비를 줄이기 위해 정점의 정보를 바탕으로 2차원 배열은 동적 할당하도록 하자.

#### 정점과 간선 정보를 이용한 인접 행렬 구현

우선 data.txt 파일의 각 줄에는 문자 'v'와 함께 정점에 대한 정보가 우선적으로 나열되어 있고, 그 이후 문자 'e'와 함께 간선에 대한 정보들이 나열되어 있다. 따라서 'v'라면 정수를 읽어 정점으로 추가하

고, 'e'라면 두 정수  $u$ 와  $v$ 를 읽어  $u$ 에서  $v$ 로 향하는 간선을 이어주면 된다. 이 때 정점 데이터에 주목을 해야 한다. 0부터 7까지의 정점 데이터가 연속적인 오름차순으로 파일에 저장되어 있지만, 중간에 6의 값을 갖는 정점이 없기 때문이다. 따라서 인접 행렬을 구성하고, 간선을 2차원 배열에 표현하고자 할 때 두 가지의 방법을 떠올려 볼 수 있다.

1) 첫째는 정수 배열을 생성하여 따로 정점 데이터들을 저장하고, 인덱스를 사용하여 인접 행렬을 표현하는 방법이다. 이 방법은 다음과 같이 정점이 정수형이 아닌 문자열인 경우, 그리고 인접 행렬의 인덱스 번호를 그대로 정점의 값으로 사용하기 어려운 경우 사용할 수 있다. 다음과 같이 자료구조2 실습 7주차 과제였던 “인접 행렬로 구현하는 지하철 노선 그래프”를 예시로 들 수 있다.



2) 두번째 방법은 따로 정점을 저장하는 배열을 사용하지 않고, 2차원 행렬의 각 행과 열의 인덱스 번호를 그대로 정점의 값으로 사용하는 것이다. 이번 문제와 같은 경우 6의 값을 제외하고 0부터 7까지 모든 정점이 존재한다. 따라서 (7x7) 행렬을 동적 할당하고, 6행과 6열은 사용을 하지 않게 되는 것이다.

이번 문제에서는 총 0부터 7까지, 6을 제외하고는 모든 정점이 존재하기 때문에 두번째 방법을 사용하는 것이 메모리의 낭비가 크지는 않다. 하지만 정점이 0의 값부터 연속적인 정수값들로만 이루어지는 그래프만을 다루지는 않게 될 것이므로, 첫번째의 방법을 사용해 그래프를 구현하는 것으로 결정하였다.

### 인접 행렬을 이용하여 구현하는 그래프

우선 그래프를 구현하기 위해서는, 다음과 같이 정수형인 정점들을 저장하기 위한 배열이 필요하다. 작성한 프로그램에서는 파일에서 읽는 정점들을 차례대로 value\_list 배열에 0번 인덱스부터 차례대로 저장하도록 하자. 다음과 같이, 정점 0부터 정점 5까지 차례대로 value\_list의 0번 인덱스부터 5번 인덱스까지, 그리고 정점 7은 value\_list의 6번 인덱스에 저장된다. 그리고 정점의 방문 여부를 확인하는 visited 배열도 선언한다. 초기값은 모두 0(FALSE)이다. 정점은 총 6개이므로, (6x6) 크기의 2차원 배열

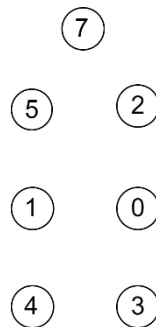
을 동적 할당하도록 하자. 행렬의 행과 열의 인덱스 번호는 각각 value\_list와 visited 배열에 접근하여 실제 정점의 값을 참조하게 된다.

	[0]	[1]	[2]	[3]	[4]	[5]	[6]
value_list	0	1	2	3	4	5	7
visited	0	0	0	0	0	0	0

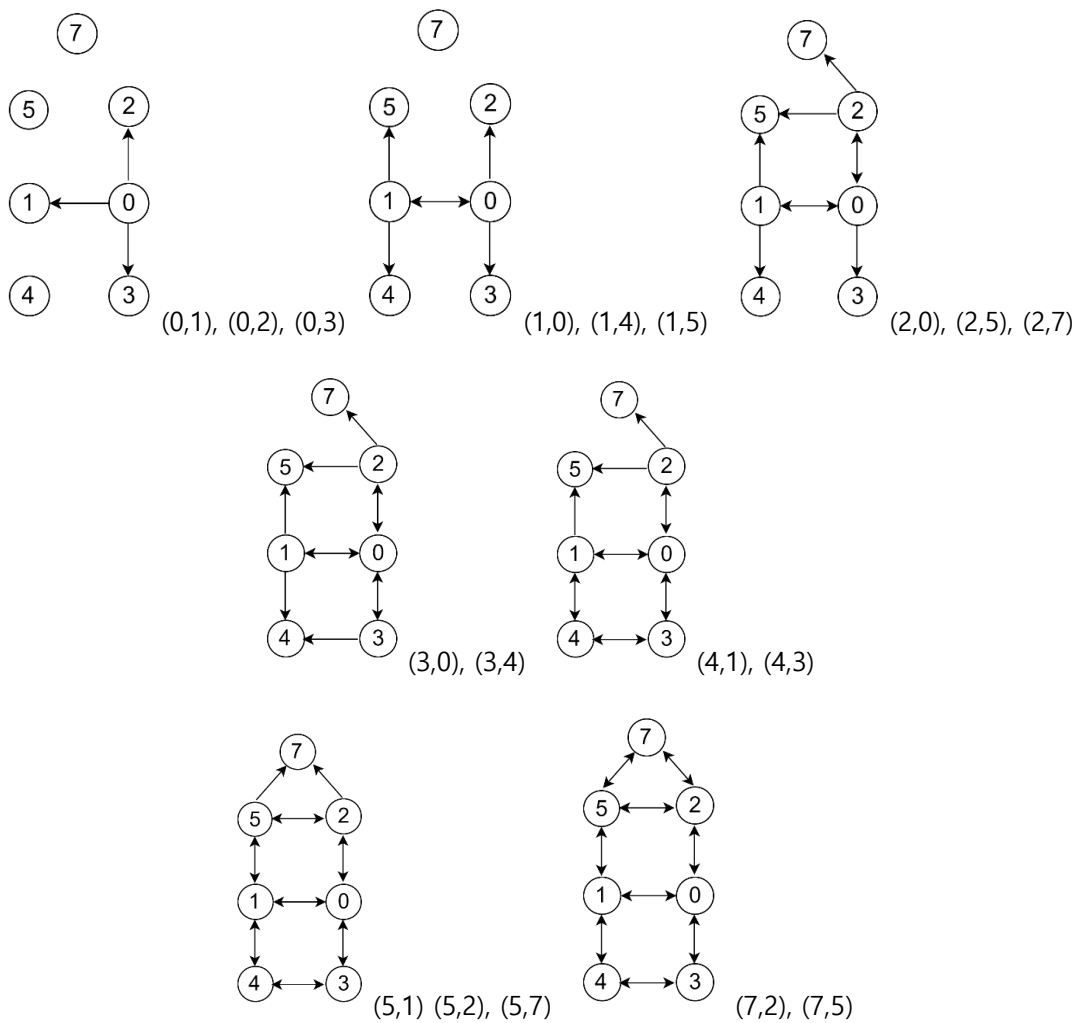
	value_list[0]	value_list[1]	value_list[2]	value_list[3]	value_list[4]	value_list[5]	value_list[6]
value_list[0]	0	0	0	0	0	0	0
value_list[1]	0	0	0	0	0	0	0
value_list[2]	0	0	0	0	0	0	0
value_list[3]	0	0	0	0	0	0	0
value_list[4]	0	0	0	0	0	0	0
value_list[5]	0	0	0	0	0	0	0
value_list[6]	0	0	0	0	0	0	0

### 1.1.1 그래프 생성 과정

우선 문자 'v'와 함께 저장되어 있는 정점 데이터들을 읽어 정점을 삽입하고, 'e'와 함께 저장되어 있는 두 정수 u와 v를 읽어 u에서 v로 향하는 간선을 삽입하는 과정을 그림으로 나타내면 다음과 같다.



위 그림은 간선을 삽입하기 이전의 그래프이다. 아래 그림은 정점 0부터 차례대로, 다른 정점으로 향하는 간선을 삽입한 과정을 나타낸 것이다.

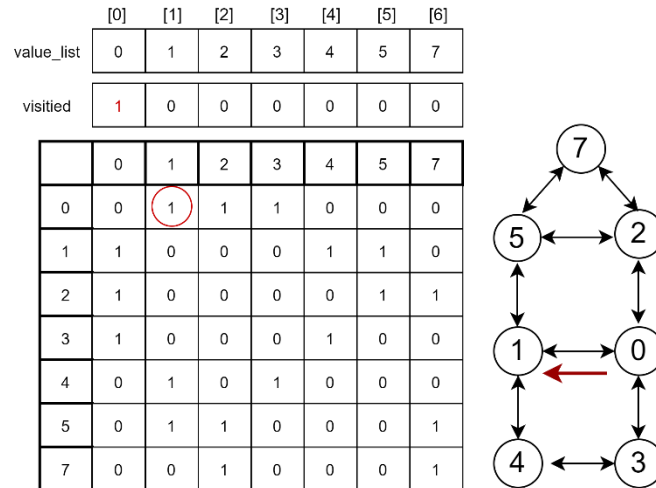


간선의 삽입이 끝나고 난 후, 인접 행렬의 결과는 다음과 같다. 그래프는 무 방향 그래프이므로, 대칭 행렬의 결과 가 나타남을 확인할 수 있다.

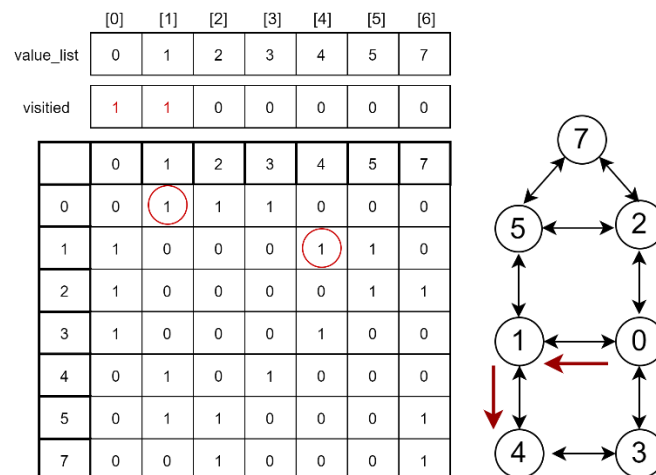
	value_ list[0]	value_ list[1]	value_ list[2]	value_ list[3]	value_ list[4]	value_ list[5]	value_ list[6]
value_ list[0]	0	1	1	1	0	0	0
value_ list[1]	1	0	0	0	1	1	0
value_ list[2]	1	0	0	0	0	1	1
value_ list[3]	1	0	0	0	1	0	0
value_ list[4]	0	1	0	1	0	0	0
value_ list[5]	0	1	1	0	0	0	1
value_ list[6]	0	0	1	0	0	1	0

### 1.1.2 깊이 우선 탐색 과정 그림

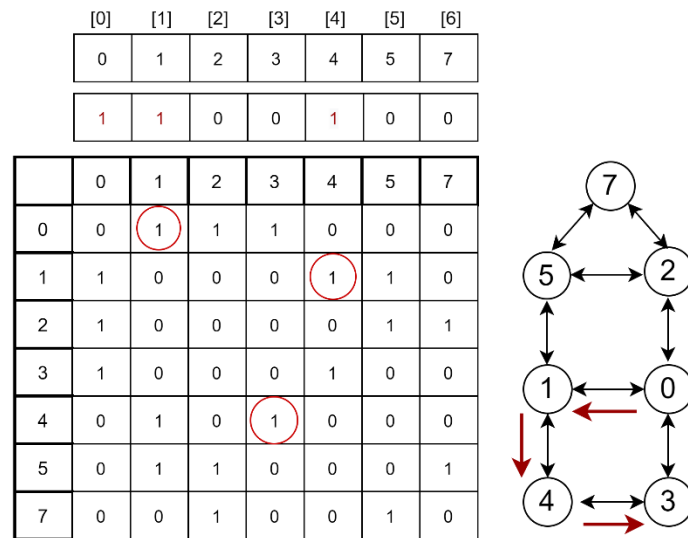
시작 정점인  $v$  에서 출발하여,  $v$  에 인접한 정점들 중 아직 방문하지 않은 정점  $u$  를 선택하여 방문하고, 정점  $u$  에서 다시 인접한 정점들 중 아직 방문하지 않은 정점을 선택하여 방문하며, 이 과정을 반복하는 깊이 우선 탐색(DFS)을 하는 과정을 아래와 같이 나타냈다.



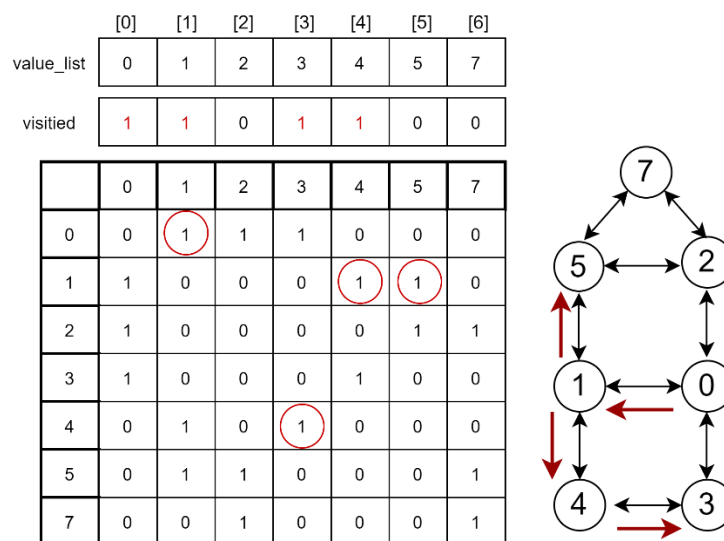
1. 시작 정점인 0(value\_list[0], visited[0])에서 깊이 우선 탐색을 시작한다. visited[0]의 값을 1(TRUE)로 변경한다. 인접한 정점들 중 아직 방문하지 않은 정점은 1(value\_list[1]) 2(value\_list[2], 3(value\_list[3])이다. 이들 중 value\_list 의 가장 작은 인덱스인 1 번에 저장되어 있는 정점 1 에 방문한다.



2. 1(value\_list[1], visited[1])에서 다시 깊이 우선 탐색을 진행한다. visited[1]의 값을 1 로 변경한다. 다음으로 정점 4 에 방문한다.

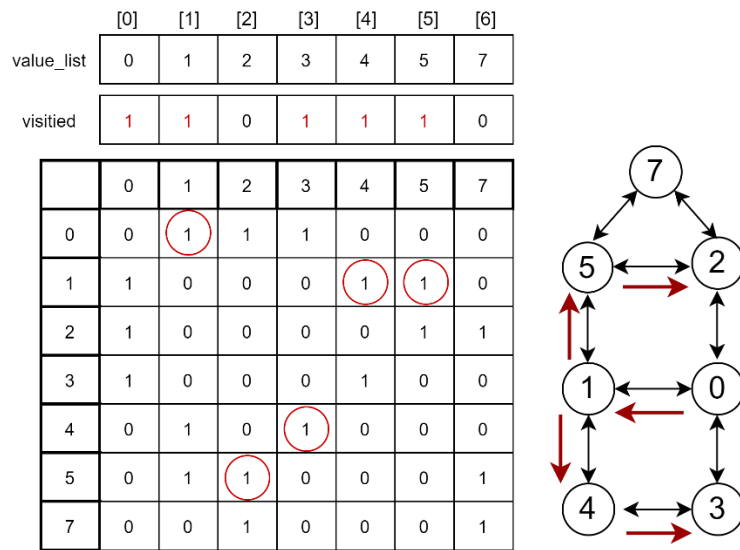


3. 4(value\_list[4], visited[4])에서 깊이 우선 탐색을 진행한다. visited[4]의 값을 변경한다. 다음으로 정점 3에 방문한다.

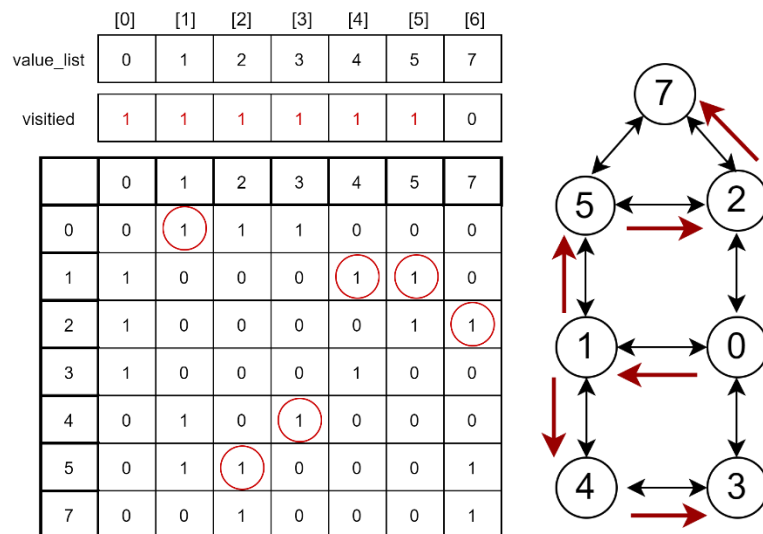


4. 3(value\_list[3], visited[3])에서 깊이 우선 탐색을 진행한다. visited[3]의 값을 변경한다. 정점 3과 인접한 정점들 중에서는, 방문하지 않은 노드들이 없기 때문에 정점 4로 돌아온다. 정점 4도 마찬가지로, 다시 정점 1에 대해 다음 방문 노드를 확인한다. 정점 1의 인접한 정점 5가 아직 방문되지 않았으므로, 정점 5를 방문한다.

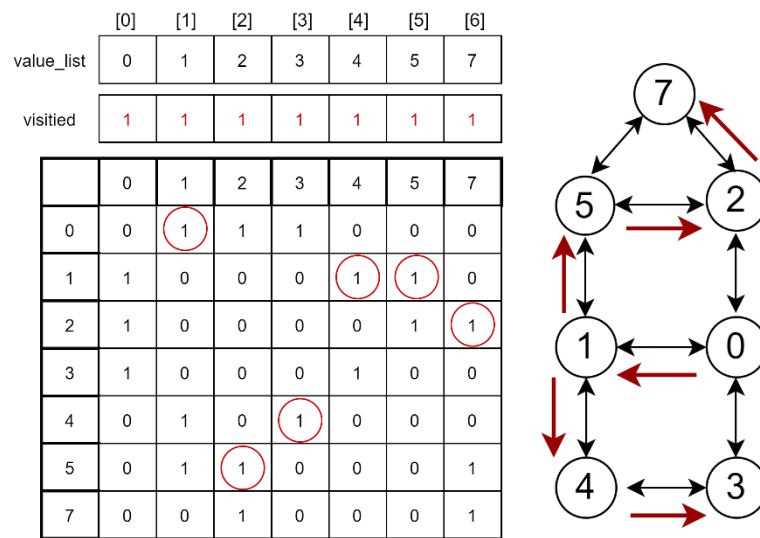




5. 5(value\_list[5], visited[5])에서 깊이 우선 탐색을 진행한다. visited[5]의 값을 변경한다. 다음으로 정점 2에 방문한다.



6. 2(value\_list[2], visited[2])에서 깊이 우선 탐색을 진행한다. visited[2]의 값을 변경한다. 다음으로 정점 7에 방문한다.



7. 7(value\_list[6], visited[6])에서 깊이 우선 탐색을 진행한다. visited[6]의 값을 변경한다. 정점 7 과 인접한 정점들 중에서는, 방문하지 않은 노드가 없고, 나머지 노드들에 대해서도 인접한 정점들 중 방문하지 않은 노드가 없으므로 깊이 우선 탐색을 종료한다.

정점의 방문 순서는 0, 1, 4, 3, 5, 2, 7이다.

## 1.2 소스 코드

```

1  /*
2  → 작성자: ·이예빈 (20204059)
3  → 작성일: ·2021.10.28
4  → 프로그램명: ·파일에서 정점과 간선의 정보를 이용하여, 인접 행렬을 통해 구현한 그래프를 구성하고,
5  → → → 이 그래프를 깊이 우선 탐색하여 출력하는 프로그램
6  */
7
8  #include <stdio.h>
9  #include <stdlib.h>
10
11  // 인접 리스트를 통해 구현하는 그래프
12  typedef struct GraphType {
13  → int n;
14  → int** adj_mat; // 인접 행렬
15  } GraphType;
16
17  // 그래프 초기화 (배열 동적 할당)
18  void init(GraphType* g, int size) {
19  → int r, c;
20  → g->n = size;
21
22  → g->adj_mat = (int**) malloc(sizeof(int*) * size); // 포인터 배열 동적 할당
23  → for (r = 0; r < size; r++) {
24  → → g->adj_mat[r] = (int*) malloc(sizeof(int) * size); // 정수형 공간 동적 할당
25  → → for (c = 0; c < size; c++)
26  → → → g->adj_mat[r][c] = 0; // 행렬의 모든 값을 0으로 초기화
27  → → }
28  }
29
30  // 간선 (u, v) 삽입 연산. u와 v는 정점이 저장되어 있는 인덱스 번호
31  void insert_edge(GraphType* g, int u, int v) {
32  → if (u >= g->n || v >= g->n) {
33  → → fprintf(stderr, "그래프: 정점 번호 오류");
34  → → return;
35  → → }
36  → g->adj_mat[u][v] = 1; // 0의 값을 1로 변경
37  }
38
39  #define TRUE 1
40  #define FALSE 0
41  #define MAX_VERTICES 30
42  int visited[MAX_VERTICES]; // 초기값: 0 (FALSE)
43
44  // 인접 행렬로 표현된 그래프에 대한 깊이 우선 탐색
45  void dfs_mat(GraphType* g, int vertex_list[], int v) {
46  → int w;
47  → visited[v] = TRUE; // 정점 v의 방문 표시
48  → printf("%d", vertex_list[v]);
49
50  → for (w = 0; w < g->n; w++)
51  → → // v에 대해 w가 인접 정점이고, w가 방문하지 않은 노드의 경우
52  → → if (g->adj_mat[v][w] && !visited[w])
53  → → → dfs_mat(g, vertex_list, w); // 정점 w에서 DFS 시작 (순환)
54  }
55
56  // 정점의 번호를 저장해놓은 배열에서 해당 정점의 인덱스 값 반환
57  int findIndex(int v, int vertex_list[], int size) {
58  → int i = 0;
59  → for (i = 0; i < size; i++)
60  → → if (v == vertex_list[i]) return i;
61  }
62
63  // 그래프의 모든 메모리 해제
64  void destroy_graph(GraphType* g) {
65  → int r;
66  → for (r = 0; r < g->n; r++)
67  → → free(g->adj_mat[r]);
68  }
69

```

```

70 int main() {
71     FILE* fp;
72     GraphType* g;
73     int vertex_list[20], size = 0; // 정점의 리스트, 정점 리스트의 요소 개수
74     char ch; // 파일에서 읽을 문자
75     int v, u, i; // 두 정점
76
77     g = (GraphType*) malloc(sizeof(GraphType));
78
79     fp = fopen("data.txt", "rt");
80     if (fp == NULL) {
81         fprintf(stderr, "파일 열기 오류");
82         exit(1);
83     }
84
85     while (!feof(fp)) {
86         fscanf(fp, "%c", &ch);
87         // v라면 정점 삽입
88         if (ch == 'v') {
89             fscanf(fp, "%d", &v);
90             vertex_list[size++] = v;
91         }
92         else if (ch == 'e') break;
93     }
94
95     init(g, size);
96
97     // 이어서 파일을 읽으며 간선 삽입
98     while (!feof(fp)) {
99         // e라면 간선 삽입
100         if (ch == 'e') {
101             fscanf(fp, "%d %d", &v, &u);
102             v = findIndex(v, vertex_list, size);
103             u = findIndex(u, vertex_list, size);
104             insert_edge(g, v, u); // 정점 v에서 정점 u로의 간선 삽입
105         }
106         fscanf(fp, "%c", &ch);
107     }
108
109     printf("-< 그래프 깊이 우선 탐색 결과 >- \n");
110     dfs_mat(g, vertex_list, 0); // 깊이 우선 탐색 0번 인덱스의 정점부터 시작
111     printf("> \n");
112
113     destroy_graph(g); // 인접 행렬의 메모리 해제
114     free(g); // 그래프 메모리 해제
115
116     return 0;
117 }

```

### 1.3 소스 코드 분석

```

/*
 * 작성자: ·이예빈 (20204059)
 * 작성일: ·2021.10.28
 * 프로그램명: ·파일에서 정점과 간선의 정보를 이용하여 인접 행렬을 통해 구현한 그래프를 구성하고,
 *           → 이 그래프를 깊이 우선 탐색하여 출력하는 프로그램
 */

#include <stdio.h>
#include <stdlib.h>

```

1. 작성자, 작성일, 프로그램명을 주석에 작성하고, 필요한 헤더 파일을 추가한다. 이 프로그램은 파일에서 정점과 간선의 정보를 읽어, 인접 행렬을 통해 구현한 그래프를 구성하고, 이 그래프를 깊이 우선 탐색(DFS, Depth First Search)하여 방문되는 정점들을 차례대로 출력하는 프로그램이다.

main 함수의 동작을 살펴보기 전, 프로그램에서 사용되는 함수들을 먼저 알아보자.

```

// 그래프 초기화 (배열 동적 할당)
void init(GraphType* g, int size) {
    int r, c;
    g->n = size;

    g->adj_mat = (int**) malloc(sizeof(int*) * size); // 포인터 배열 동적 할당
    for (r = 0; r < size; r++) {
        g->adj_mat[r] = (int*) malloc(sizeof(int) * size); // 정수형 공간 동적 할당
        for (c = 0; c < size; c++)
            g->adj_mat[r][c] = 0; // 행렬의 모든 값을 0으로 초기화
    }
}

```

2. 그래프의 포인터 g 와 그래프의 정점 개수인 size 변수를 매개 변수로 받아 인접 행렬을 동적 할당하고, 그래프를 초기화하는 init 함수이다. 메모리 동적 할당을 위해 반복문의 제어 변수 r 과 c 를 선언한다. 포인터 g 가 가리키는 GraphType 구조체의 n 멤버에는 size 의 값을 대입한다.

3. 구조체의 adj\_mat 이중 포인터에 정수형 포인터 배열 공간을 size 개수만큼 동적 할당한다. 각 포인터 배열 공간에는 size 개수만큼 정수 공간을 동적 할당하고, 2 차원 배열의 모든 값은 0 으로 초기화한다.

```

// 간선 (u, v) 삽입 연산 -- u와 v는 정점이 저장되어 있는 인덱스 번호
void insert_edge(GraphType* g, int u, int v) {
    if (u >= g->n || v >= g->n) {
        fprintf(stderr, "그래프: 정점 번호 오류");
        return;
    }
    g->adj_mat[u][v] = 1; // 0의 값을 1로 변경
}

```

4. 간선을 삽입하는 insert\_edge 함수이다. 그래프를 가리키는 포인터 g 와 간선의 시작 정점 u 와 끝 정점 v 를 매개 변수로 전달 받아 두 정점에 대해 방향을 갖는 간선을 삽입하도록 한다. 우선적으로 u 와 v 의 값이 g 의 n 멤버보다 큰 경우에는 정점 번호 오류가 났다는 출력문과 함께 함수를 종료하도록 한다. 그렇지 않은 경우, 0 으로 초기화되어 있었던 2 차원 배열의 [u][v]의 값을 1 로 변경한다.

다음은 깊이 우선 탐색에 필요한 define 문, 배열, 함수가 정의된 부분이다. 자세히 살펴보자.

```
#define TRUE 1
#define FALSE 0
#define MAX_VERTICES 30
int visited[MAX_VERTICES]; // 초기값: 0 (FALSE)
```

5. define 문을 사용해 TRUE 와 FALSE 상수를 각각 1 과 0 의 값으로 정의한다. 노드의 방문 여부를 확인하는 정수 배열 visited 를 30 의 값으로 정의된 MAX\_VERTICES 크기 만큼 선언한다. 전역으로 선언된 배열이므로 값은 모두 0(FALSE)으로 초기화된 상태이다.

```
// 인접 행렬로 표현된 그래프에 대한 깊이 우선 탐색
void dfs_mat(GraphType* g, int vertex_list[], int v) {
    int w;
    visited[v] = TRUE; // 정점 v 의 방문 표시
    printf("%d", vertex_list[v]);
```

6. 인접 행렬로 표현된 그래프에 대해 순환적인 방법으로 깊이 우선 탐색을 하는 dfs\_mat 함수이다. 그래프 구조체를 가리키는 포인터 g, 정점의 값들을 저장하고 있는 vertex\_list 배열, 그리고 깊이 우선 탐색을 시작할 정점 v 를 매개 변수로 전달 받는다. (이때, 우리는 vertex\_list 배열을 통해 실제 정점 값에 접근하기 때문에, v 는 사실상 정점의 값이 아닌 vertex\_list 의 인덱스 번호이다.) 우선, 다음으로 방문하게 될 정점인 w 변수를 선언한다. 현재 방문한 v 에 대해서는 visited[v]의 값을 TRUE 로 변경하고, vertex\_list 에 참조하여 v 번 인덱스에 해당하는 값을 출력한다.

```
    for (w = 0; w < g->n; w++)
        // v 에 대해 w 가 인접 정점이고, w 가 방문하지 않은 노드의 경우
        if (g->adj_mat[v][w] && !visited[w])
            dfs_mat(g, vertex_list, w); // 정점 w 에서 DFS 시작 (순환)
}
```

7. 이후, 0 부터 g->n 이전까지 반복문이 돌며, v 에 인접한 정점들 중 (adj\_mat[v][w]==1), 아직 방문하지 않은 노드라면 (visited[w] == FALSE), w 를 다음 정점으로 하여 dfs\_mat 함수를 순환 호출하여 깊이 우선 탐색을 반복하도록 한다.

```

// 정점의 번호를 저장해놓은 배열에서 해당 정점의 인덱스 값을 반환
int findIndex(int v, int vertex_list[], int size) {
    int i = 0;
    for (i = 0; i < size; i++)
        if (v == vertex_list[i]) return i;
}

```

8. 정점의 값들을 저장해놓은 vertex\_list 배열에서 정점 v가 저장되어 있는 인덱스 번호를 반환하는 findIndex 함수이다.

```

// 그래프의 모든 메모리 해제
void destroy_graph(GraphType* g) {
    int r;
    for (r = 0; r < g->n; r++)
        free(g->adj_mat[r]);
}

```

9. 인접 행렬의 모든 메모리를 해제하는 함수이다. 각 포인터 배열에 할당된 정수 공간들을 해제한다.

다음으로 main 함수의 동작을 살펴보자.

```

int main() {
    FILE* fp;
    GraphType* g;
    int vertex_list[20], size = 0; // 정점의 리스트, 정점 리스트의 요소 개수
    char ch; // 파일에서 읽을 문자
    int v, u; // 두 정점에 대한 인덱스 번호

    g = (GraphType*) malloc(sizeof(GraphType));
}

```

10. 파일 포인터 fp, 그래프의 구조체 g를 선언한다. 파일로부터 정점들을 읽어 차례대로 정수형 정점 값들을 최대 20개 저장할 수 있는 vertex\_list 배열을 선언한다. vertex\_list 배열에 실제로 저장되는 정점의 개수를 셀 size 변수를 0으로 초기화한다. 파일의 각 줄의 처음에 저장되어 있는 문자를 읽을 ch, 그리고 정점의 값들 및 정점에 대한 배열의 인덱스 번호를 저장할 정수형 v와 u를 선언한다.

11. 포인터 g에 GraphType 형 구조체 공간을 동적 할당한다.

```

→ fp=fopen("data.txt","rt");
→ if(fp==NULL){
→     fprintf(stderr,"파일 열기 오류");
→     exit(1);
→ }

→ while(!feof(fp)){
→     fscanf(fp,"%c",&ch);
→     //v라면 정점 삽입
→     if(ch=='v'){
→         fscanf(fp,"%d",&v);
→         vertex_list[size++] = v;
→     }
→     else if(ch=='e') break;
→ }

→ init(g, size);

```

12. data.txt 파일을 읽기 모드로 연다.

13. 파일의 끝을 가리킬 때까지 읽는다. 먼저 각 줄의 처음에 저장되어 있는 문자를 읽도록 한다. 'v'를 읽으면 다시 정수 변수를 하나 읽어, 해당 정점을 vertex\_list 에 삽입한다. size 가 0 부터 차례대로 증가하며, 파일에 저장되어 있는 값들은 0 번 인덱스부터 vertex\_list 배열에 차례대로 저장된다.

14. 만약 'e' 문자가 나타나면 반복문은 종료한다. 파일은 'v'에 대한 값들이 모두 나온 후에 간선의 정보가 저장되어 있는 'e' 문자가 나타나므로, 파일을 두번째로 읽기 전, size 를 두번째 인수로 전달하여 init 함수를 호출하고, 그래프의 인접 행렬을 동적 할당하도록 한다.

```

→ //이어서 파일을 읽으며 간선 삽입
→ while(!feof(fp)){
→     //e라면 간선 삽입
→     if(ch=='e'){
→         fscanf(fp,"%d%d",&v,&u);
→         v=findIndex(v,vertex_list,size);
→         u=findIndex(u,vertex_list,size);
→         insert_edge(g,v,u); //정점 v에서 정점 u로의 간선 삽입
→     }
→     fscanf(fp,"%c",&ch);
→ }

```

15. 이어서 파일을 읽으며 간선을 삽입한다. 파일로부터 두 정수를 v 와 u 변수로 읽고, findIndex 함수를 호출하여 vertex\_list 배열에서 v 와 u 정점이 저장되어 있는 인덱스 번호를 v 와 u 에 각각 다시 대입한다.

16. insert\_edge 함수를 호출하여, 정점 v(인덱스 번호)에서 정점 u(인덱스 번호)로 향하는 간선을 삽입한다.



```
→ printf("-<그래프.깊이.우선.탐색.결과>-<\n<");
→ dfs_mat(g, vertex_list, 0); → // 깊이.우선.탐색.0번.인덱스의.정점부터.시작
→ printf(">\n");
```

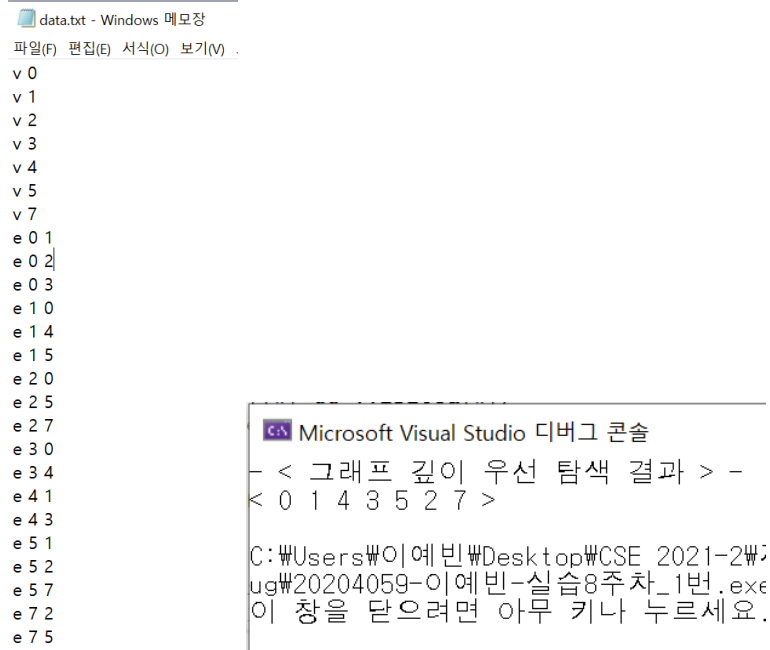
17. 0 번 인덱스에 저장되어 있는 정점(정점 0)부터 시작하여 깊이 우선 탐색을 진행한다. dfs\_mat 함수를 호출한다.

```
→ destroy_graph(g); → // 인접.행렬의.메모리.해제
→ free(g); → → → // 그래프.메모리.해제
→ return 0;
}□
```

18. 마지막으로 destroy\_graph 를 호출하여 인접 행렬의 메모리를 해제하고, 그래프의 메모리 또한 해제하여 전체 프로그램을 종료한다.

## 1.4 실행창

data.txt 파일은 다음과 같이 정점 정보(문자 'v'와 정수의 정점 데이터)와 간선 정보(문자 'e'와 두 정점 데이터)가 나타나 있다.



The image shows two windows. The top window is a Notepad editor titled 'data.txt - Windows 메모장' containing the following text:

```

파일(F) 편집(E) 서식(O) 보기(V)
v 0
v 1
v 2
v 3
v 4
v 5
v 7
e 0 1
e 0 2
e 0 3
e 1 0
e 1 4
e 1 5
e 2 0
e 2 5
e 2 7
e 3 0
e 3 4
e 4 1
e 4 3
e 5 1
e 5 2
e 5 7
e 7 2
e 7 5

```

The bottom window is the 'Microsoft Visual Studio 디버그 콘솔' (Debug Console) showing the output of a DFS algorithm:

```

- < 그래프 깊이 우선 탐색 결과 > -
< 0 1 4 3 5 2 7 >

C:\Users\이예빈\Desktop\CSE 2021-2학기\20204059-이예빈-실습8주차_1번.exe
이 창을 닫으려면 아무 키나 누르세요.

```

결과창은 다음과 같이 그래프의 깊이 우선을 탐색한 결과가 나타난다. 방문한 정점의 순서대로 출력된 것을 확인할 수 있다.

아래는 코드를 일부 추가하여 정점 0이 아닌 다른 정점부터 깊이 우선 탐색을 시작할 수 있도록 해보았다.

```

→ printf("정점·몇·부터·깊이·우선·탐색을·시작하겠습니까?·:·");
→ scanf("%d", &i);
→ printf("--<·그래프·깊이·우선·탐색·결과·>·--·\n<");
→ dfs_mat(g, vertex_list, findIndex(i, vertex_list, size)); → //·깊이·우선·탐색·정점·i·부터·시작
→ printf(">\n");

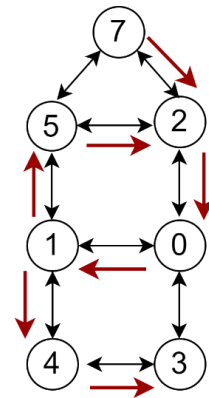
```

아래는 직접 입력을 받아, 정점 7 과 정점 4 부터 깊이 우선 탐색을 하여 방문한 정점들이 올바르게 출력된 결과창이다. 깊이 우선 탐색을 정확하게 한 것을 확인할 수 있다,

Microsoft Visual Studio 디버그 콘솔

```

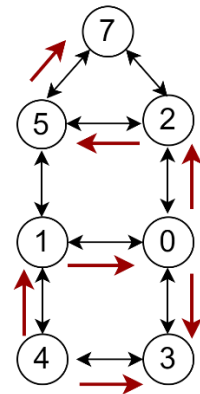
정점 몇 부터 깊이 우선 탐색을 시작하겠습니까? : 7
- < 그래프 깊이 우선 탐색 결과 > -
< 7 2 0 1 4 3 5 >
    
```



Microsoft Visual Studio 디버그 콘솔

```

정점 몇 부터 깊이 우선 탐색을 시작하겠습니까? : 4
- < 그래프 깊이 우선 탐색 결과 > -
< 4 1 0 2 5 7 3 >
    
```



### 1.5 느낀점

이 문제를 풀면서 크게 두 가지 고민을 했던 것 같다. 첫번째는 인접 행렬과 인접 리스트 중 어떠한 방법으로 그래프를 구현할 것이냐 였다. 인접 행렬로 그래프를 구현하게 되는 경우에는 6의 값을 갖는 정점이 없기 때문에 그 이후에 해결해야 할 또다른 문제점이 생긴다는 점에서, 처음에는 인접 리스트로 그래프를 구현하고자 했다. 그러나, 인접 리스트와 같은 경우 각 정점을 데이터 필드로 하는 노드를 리스트의 처음에 삽입하는 순서로 인해 인접한 정점들의 순서 또한 바뀌므로 예시 실행창과 동일한 결과를 얻기는 쉽지 않다고 판단하였다. 따라서 인접 행렬으로 그래프를 구현하기로 결정하였다. 그 이후에는 6이라는 값을 어떻게 처리하는게 좋을지에 대한 고민이 있었다. 수업 시간에 최대값인 정점 7을 기준으로 (7x7)크기의 행렬을 할당한 후, 6번 행과 열을 사용하지 않는 방법을 말씀하신 것으로 언뜻 기억하는데, 이 방법을 사용하면 더 많은 수의 정점이 사용되지 않는 경우 메모리의 낭비가 크다는 문제와 이전 7주차 과제와 같이 정수가 아닌 정점의 경우에는 정점들을 저장하는 배열을 꼭 사용해야 해결된다는 점에서 정점들을 따로 배열에 저장해놓고, 정점이 저장되어 있는 배열의 인덱스 값을 정점을 대신하여 그래프를 구성하는 것이 가장 적합하다고 판단하게 되었다. 여러가지 시행착오를 겪었지만, 이를 통해 인접 리스트는 리스트에 노드를 삽입하는 순서가 있기 때문에 원하는 결과를 동일하게 도출하는 문제에서는 구현에 있어서 쉽지 않다는 것을 다시 한 번 깨닫게 된 것 같다.

## 2. 그래프의 너비 우선 탐색(BFS, Breadth First Search)

**너비 우선 탐색 프로그램**

■ 그래프 너비 우선 탐색 프로그램

- data.txt에서 데이터를 읽어와 프로그램을 작성하시오. 이때 v는 정점, e는 간선을 의미한다. 프로그램의 실행과정은 아래의 그림과 같다.

(a) 큐 0      (b) 큐      (c) 큐 1

(d) 큐 1 2      (e) 큐 1 2 4

```

data1.txt - 0
파일(F)  편집(E)
v 0
v 1
v 2
v 3
v 4
e 3 4
e 2 4
e 2 3
e 1 2
e 0 4
e 0 2
e 0 1
        
```

C:\WINDOWS\system32\cmd.exe

너비 우선 탐색 : 0 1 2 4 3

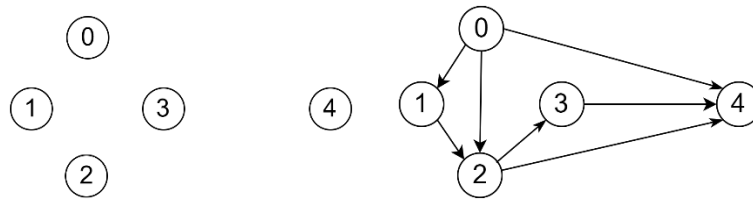
계속하려면 아무 키나 누르십시오 . . .

### 2.1 문제 분석

2번 문제는 1번 문제와 동일한 방법으로 파일로부터 정점과 간선의 정보를 읽어 그래프를 생성하고, 이 그래프를 너비 우선 탐색(BFS, Breadth First Search)하며 방문한 정점들을 출력하는 문제이다. 1번 문제와 파일에 저장되어 있는 데이터들 비교해보면, 정점이 0번부터 차례대로 4까지 중간에 비어있는 값 없이 연속으로 정점의 값이 주어졌다는 것, 그리고 간선의 정보를 통해 그래프는 방향 그래프라는 것을 알 수 있으며, 1번 문제의 그래프와 차이가 있음을 확인할 수 있다. 1번 문제와 동일하게 인접 행렬을 이용하여 그래프를 구현하도록 하자.

#### 그래프와 인접 행렬

아래의 왼쪽 그림은, 아직 간선을 연결하지 않은 정점은 다음과 같이 0부터 4까지의 정점들이, 오른쪽 그림은 간선을 추가하여 완성한 그래프의 그림이다.



이를 인접 행렬로 표현하면 다음과 같다. 1 번 문제와 동일한 방법으로 그래프를 생성하였다. 1 번과 동일하게 value\_list 와 visited 배열 또한 생성한다. 정점 0 부터 정점 4 까지 중간에 비어 있는 값이 없으므로, 행렬의 행과 열의 인덱스 값을 그대로 정점의 값으로 사용해도 되지만, 이번에도 1 번과 문제와 동일하게 정점의 값이 아닌, value\_list 배열에 저장한 정점의 위치인 인덱스 값을 이용하여 탐색 방법을 사용하도록 하자.

	[0]	[1]	[2]	[3]	[4]
value_list	0	1	2	3	4
visited	0	0	0	0	0

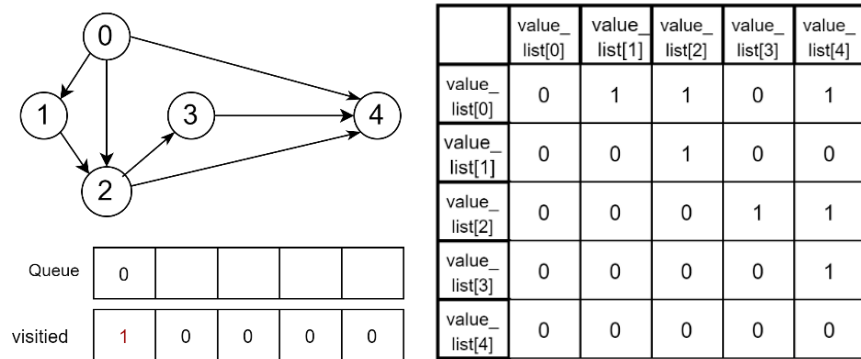
  

	value_ list[0]	value_ list[1]	value_ list[2]	value_ list[3]	value_ list[4]
value_ list[0]	0	1	1	0	1
value_ list[1]	0	0	1	0	0
value_ list[2]	0	0	0	1	1
value_ list[3]	0	0	0	0	1
value_ list[4]	0	0	0	0	0

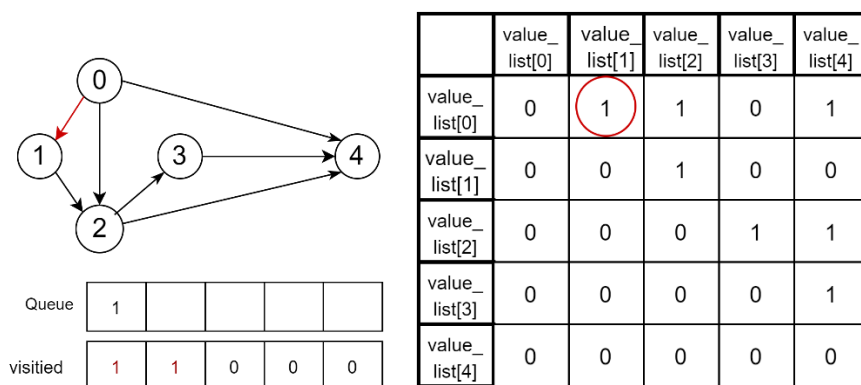
### 2.1.1 너비 우선 탐색 과정 그림

그래프의 너비 우선 탐색(BFS, Breadth First Search)은 원형 큐 구조를 이용하여 방문한 노드의 인접한 정점들을 모두 삽입하고, 큐에 삽입된 정점들을 차례대로 다시 꺼내어 아직 방문되지 않은 정점에 대해 다시 너비 우선 탐색을 하는 것을 반복하게 된다.

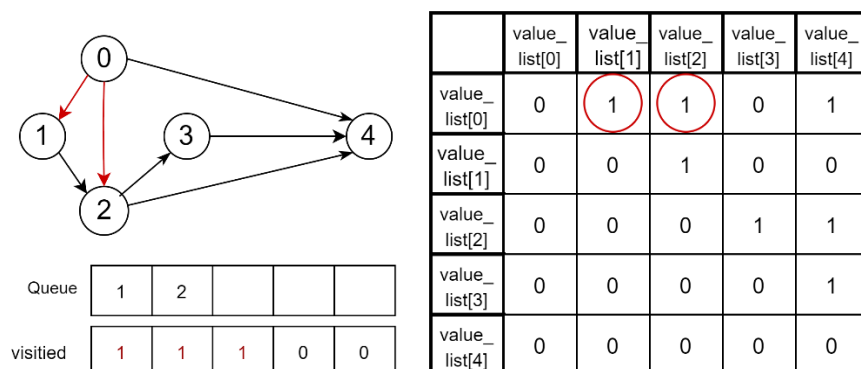
이번 문제에서 너비 우선 탐색을 하는 과정을 다음 그림을 통해 자세히 살펴보자.



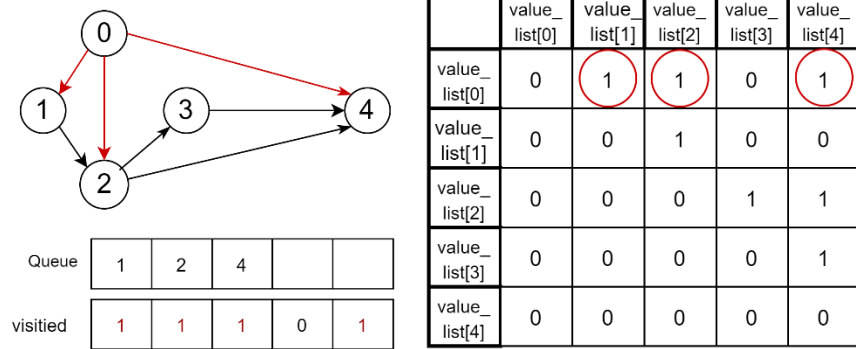
1. 우선 정점 0(value\_list[0])부터 너비 우선 탐색을 시작한다. 정점 0 을 큐에 삽입하고, visited[0]의 값을 변경한다.



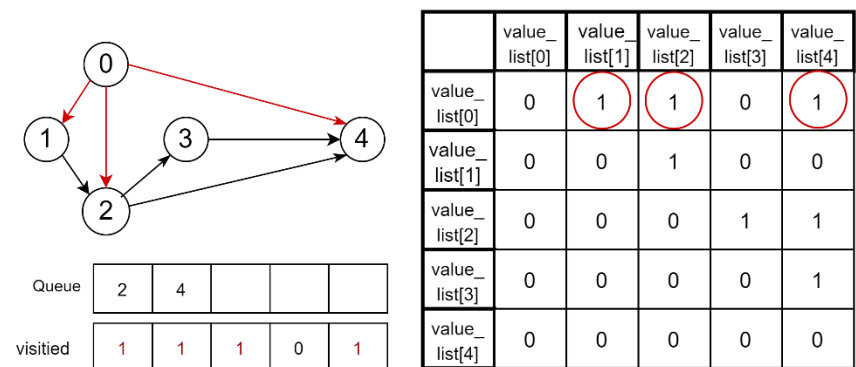
2. 정점 0 을 큐에서 삭제함과 동시에, 0 과 인접한 정점들 중 방문하지 않은 정점들을 모두 큐에 차례대로 삽입한다. 정점 1 을 삽입하고, visited[1]의 값을 변경한다.



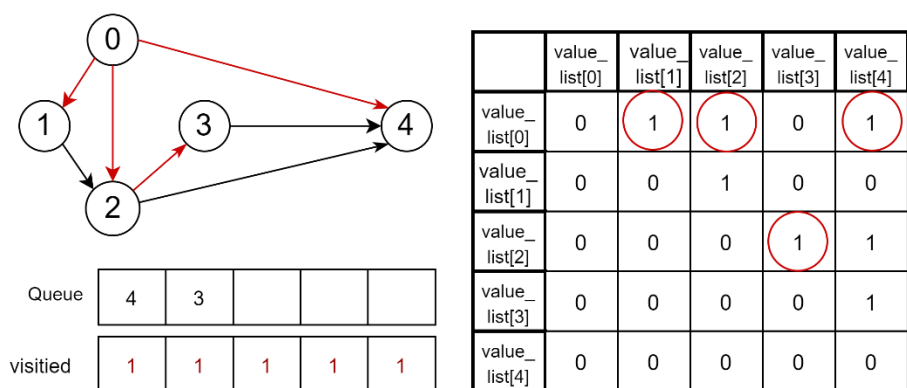
3. 정점 2 를 삽입한다. visited[2]의 값을 변경한다.



3. 정점 4 를 삽입한다. visited[4]의 값을 변경한다.



4. 더 이상 0 과 인접하며, 방문하지 않은 정점이 존재하지 않는다. 따라서 큐에서 가장 앞에 위치해있던 정점 1 을 삭제하고, 삭제함과 동시에 인접한 정점들 중 방문하지 않은 정점이 있는지 확인한다. 그러한 정점이 없으므로 다음 단계로 넘어간다.



5. 다음으로 큐에서 2 를 삭제하고, 인접한 정점들 중 아직 방문하지 않은 정점 3 을 큐에 삽입한다. visited[3]의 값을 변경한다.



2021/10/28

6. 이후, 4 와 3 에 대해 위의 과정들은 반복한다. 더 이상 인접한 정점들 중 방문하지 않은 노드가 없으므로 너비 우선 탐색은 종료한다.

방문의 순서는 0, 1, 4, 3 이다.

## 2.2 소스 코드

```

1  /*
2  → 작성자: ·이예빈 (20204059)
3  → 작성일: ·2021.10.28
4  → 프로그래밍: ·파일에서 ·점점과 ·간선의 ·정보를 ·이용하여 ·그래프를 ·구성하고,
5  →           →           이 ·그래프를 ·너비 ·우선 ·탐색하여 ·출력하는 ·프로그램
6  */
7
8  #include <stdio.h>
9  #include <stdlib.h>
10
11  // ·인접 ·행렬을 ·통해 ·구현하는 ·그래프
12  typedef struct GraphType {
13  →   int n;
14  →   int** adj_mat; // ·인접 ·행렬
15  } GraphType;
16
17  // ·큐 ·구현
18  typedef int element;
19  #define MAX_QUEUE_SIZE 10
20
21  typedef struct QueueType {
22  →   element queue[MAX_QUEUE_SIZE];
23  →   int front, rear;
24  } QueueType;
25
26  void init_queue(QueueType* q) {
27  →   q->front = q->rear = 0;
28  }
29
30  int is_empty(QueueType* q) { return q->front == q->rear; }
31  int is_full(QueueType* q) { return ((q->rear + 1) % MAX_QUEUE_SIZE == q->front); }
32
33  void enqueue(QueueType* q, element item) {
34  →   if (is_full(q)) {
35  →       fprintf(stderr, "큐가 ·포화 ·상태 ·\n");
36  →       exit(1);
37  →   }
38  →   q->rear = (q->rear + 1) % MAX_QUEUE_SIZE;
39  →   q->queue[q->rear] = item;
40  }
41
42  element dequeue(QueueType* q) {
43  →   if (is_empty(q)) {
44  →       fprintf(stderr, "큐가 ·공백 ·상태 ·\n");
45  →       exit(1);
46  →   }
47  →   q->front = (q->front + 1) % MAX_QUEUE_SIZE;
48  →   return q->queue[q->front];
49  }

```

```

47 //그래프 초기화 (배열 동적 할당)
48 void init(GraphType *g, int size) {
49     int r, c;
50     g->n = size;
51
52     g->adj_mat = (int**) malloc(sizeof(int*) * size); //포인터 배열 동적 할당
53     for (r = 0; r < size; r++) {
54         g->adj_mat[r] = (int*) malloc(sizeof(int) * size); //정수형 공간 동적 할당
55         for (c = 0; c < size; c++)
56             g->adj_mat[r][c] = 0; //행렬의 모든 값을 0으로 초기화
57     }
58 }
59
60 //간선 (u, v) 삽입. 연산. u와 v는 정점이 저장되어 있는 인덱스 번호
61 void insert_edge(GraphType *g, int u, int v) {
62     if (u >= g->n || v >= g->n) {
63         fprintf(stderr, "그래프: 정점 번호 오류");
64         return;
65     }
66     g->adj_mat[u][v] = 1; //0의 값을 1로 변경
67 }
68
69 #define TRUE 1
70 #define FALSE 0
71 #define MAX_VERTICES 30
72 int visited[MAX_VERTICES];
73
74 //인접 행렬을 이용한 너비 우선 탐색의 구현
75 void bfs_mat(GraphType *g, int vertex_list[], int v) {
76     QueueType *q;
77     int w;
78
79     q = (QueueType*) malloc(sizeof(QueueType));
80     init_queue(q);
81
82     //첫 정점
83     enqueue(q, v);
84     visited[v] = TRUE;
85     printf("%d", vertex_list[v]); //첫 번째 v 인덱스의 정점 출력
86
87     while (!is_empty(q)) {
88         v = dequeue(q); //정점 v 인덱스 삽입
89         for (w = 0; w < g->n; w++)
90             //v 정점의 인접한 정점 중 방문하지 않은 정점 모두 큐에 삽입
91             if (g->adj_mat[v][w] && !visited[w]) {
92                 enqueue(q, w);
93                 visited[w] = TRUE; //삽입할 때 visited 값을 1로 변경
94                 printf("%d", vertex_list[w]); //enqueue할 때 정점 출력
95             }
96     }
97     free(q); //큐 메모리 해제
98 }
99
100 //정점의 번호를 저장해놓은 배열에서 해당 정점의 인덱스 값 반환
101 int findIndex(int v, int vertex_list[], int size) {
102     int i = 0;
103     for (i = 0; i < size; i++)
104         if (v == vertex_list[i]) return i;
105 }
106
107 //그래프의 모든 메모리 해제
108 void destroy_graph(GraphType *g) {
109     int r;
110     for (r = 0; r < g->n; r++)
111         free(g->adj_mat[r]);
112 }

```

```

113 int main() {
114     FILE* fp;
115     GraphType* g;
116     int vertex_list[20], size = 0; // 정점의 리스트, 정점 리스트의 요소 개수
117     char ch; // 파일에서 읽을 문자
118     int v, u; // 두 정점에 대한 인덱스 번호
119
120     g = (GraphType*) malloc(sizeof(GraphType));
121
122     fp = fopen("data.txt", "rt");
123     if (fp == NULL) {
124         fprintf(stderr, "파일 열기 오류");
125         exit(1);
126     }
127
128     while (!feof(fp)) {
129         fscanf(fp, "%c", &ch);
130         // v라면 정점 삽입
131         if (ch == 'v') {
132             fscanf(fp, "%d", &v);
133             vertex_list[size++] = v;
134         }
135         else if (ch == 'e') break;
136     }
137
138     init(g, size); // 그래프 초기화
139     // 이어서 파일을 읽으며 간선 삽입
140     while (!feof(fp)) {
141         // e라면 간선 삽입
142         if (ch == 'e') {
143             fscanf(fp, "%d %d", &v, &u);
144             v = findIndex(v, vertex_list, size);
145             u = findIndex(u, vertex_list, size);
146             insert_edge(g, v, u); // 정점 v에서 정점 u로의 간선 삽입
147         }
148         fscanf(fp, "%c", &ch);
149     }
150
151     printf("--< 그래프 너비 우선 탐색 결과 --> \n");
152     bfs_mat(g, vertex_list, 0); // 너비 우선 탐색 0번 인덱스의 정점부터 시작
153     printf("> \n");
154
155     destroy_graph(g); // 인접 행렬의 메모리 해제
156     free(g); // 그래프 메모리 해제
157
158     return 0;
159 }

```

## 2.3 소스 코드 분석

```

/*
→ 작성자: ·이예빈 (20204059)
→ 작성일: ·2021.10.28
→ 프로그램명: ·파일에서 정점과 간선의 정보를 이용하여 그래프를 구성하고,
→           → 이 그래프를 너비 우선 탐색하여 출력하는 프로그램
*/

#include <stdio.h>
#include <stdlib.h>

```

1. 주석에 작성자, 작성일, 프로그램명을 작성하고, 필요한 헤더 파일들을 추가한다.

(생략 – 나머지 그래프의 구조체, 인접 행렬을 동적 할당하고 그래프를 초기화하고, 정점과 간선을 삽입하는 등의 그래프를 생성하는 과정은 모두 1번의 소스 코드와 동일하므로 생략하도록 한다.)

다음은 너비 우선 탐색에서 필요한 원형 큐 구조를 구현하는데 필요한 부분이다.

```

//·큐·구현
typedef int element;
#define MAX_QUEUE_SIZE 10

typedef struct QueueType {
→   element queue[MAX_QUEUE_SIZE];
→   int front, rear;
} QueueType;

void init_queue(QueueType* q) {
→   q->front = q->rear = 0;
}

int is_empty(QueueType* q) { return q->front == q->rear; }
int is_full(QueueType* q) { return ((q->rear + 1) % MAX_QUEUE_SIZE == q->front); }

void enqueue(QueueType* q, element item) {
→   if (is_full(q)) {
→       fprintf(stderr, "큐가 포화 상태\n");
→       exit(1);
→   }
→   q->rear = (q->rear + 1) % MAX_QUEUE_SIZE;
→   q->queue[q->rear] = item;
}

element dequeue(QueueType* q) {
→   if (is_empty(q)) {
→       fprintf(stderr, "큐가 공백 상태\n");
→       exit(1);
→   }
→   q->front = (q->front + 1) % MAX_QUEUE_SIZE;
→   return q->queue[q->front];
}

```

1. 정수형 int 를 element 라는 이름으로 정의한다. 큐에 삽입할 수 있는 최대 크기 MAX\_QUEUE\_SIZE 는 10 으로 설정한다.

2. front 와 rear 을 0 으로 초기화하는 init\_queue 함수, 공백 상태와 포화 상태를 검출하는 is\_empty, is\_full 함수, 그리고 큐에 요소를 삽입하는 enqueue, 삭제하는 dequeue 함수이다.

다음으로 너비 우선 탐색을 하는 bfs\_mat 함수를 살펴보자.

```
#define TRUE 1
#define FALSE 0
#define MAX_VERTICES 30
int visited[MAX_VERTICES];
```

3. TRUE 와 FALSE 상수를 각각 1 과 0 의 값으로 정의한다. 30 의 값으로 정의한 MAX\_VERTICES 크기의 visited 배열도 선언한다. 전역으로 선언했기 때문에 초깃값은 모두 0(FALSE)이다.

```
// 인접 행렬을 이용한 너비 우선 탐색의 구현
void bfs_mat(GraphType* g, int vertex_list[], int v){
    QueueType* q;
    int w;

    q = (QueueType*) malloc(sizeof(QueueType));
    init_queue(q);
```

4. 인접 행렬로 구현된 그래프를 너비 우선 탐색하는 bfs\_mat 함수는 그래프 구조체의 포인터 g, 정점의 값들을 저장한 vertex\_list 배열, 그리고 정점(사실상 정점에 해당하는 배열의 인덱스 번호) v 를 매개 변수로 전달 받는다. QueueType 큐 구조체 포인터 q 를 선언하고, 인접한 정점의 인덱스를 의미하는 정수형 변수 w 를 선언한다.

5. q 포인터에 큐 구조체를 동적 할당하고, init\_queue 함수를 호출하여 큐를 초기화한다.

```
    // 첫 정점
    enqueue(q, v);
    visited[v] = TRUE;
    printf("%d", vertex_list[v]); // 첫번째 v 인덱스의 정점 출력
```

6. 첫 정점(정점의 인덱스)을 큐에 삽입한다. 삽입한 후에는 해당 정점에 해당하는 visited 값을 TRUE 로 변경시킨다. 그리고 vertex\_list 의 v 번 인덱스의 정점을 출력한다.

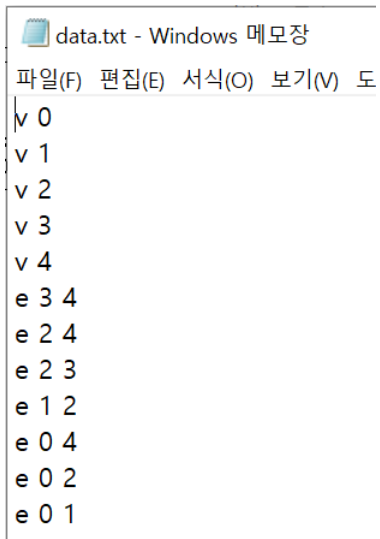
```
    while(!is_empty(q)){
        v = dequeue(q); // 정점 v 인덱스 삭제
        for(w = 0; w < g->n; w++){
            // v 정점의 인접한 정점 중 방문하지 않은 정점 모두 큐에 삽입
            if(g->adj_mat[v][w] && !visited[w]){
                enqueue(q, w);
                visited[w] = TRUE; // 삽입할 때 visited 값을 1로 변경
                printf("%d", vertex_list[w]); // enqueue 할 때 정점 출력
            }
        }
        free(q); // 큐 메모리 해제
    }
```

7. 다음으로 큐에 삽입되어 있는 정점을 삭제함과 동시에 해당 정점에 인접하면서 아직 방문하지 않은 정점들을 큐에 삽입하기를 반복하는 너비우선 탐색 과정을 진행한다. 정점을 큐에 삽입하는 것이 방문한다는 뜻이므로, 큐에 한 정점을 삽입할 때 visited 값을 변경하고, 정점을 출력하도록 한다.

(생략 – main 함수도 1 번 문제의 main 함수와 동일하다. 다만, 호출하는 함수가 dfs\_mat 함수가 아닌, 너비 우선 탐색을 하는 bfs\_mat 함수이다.)

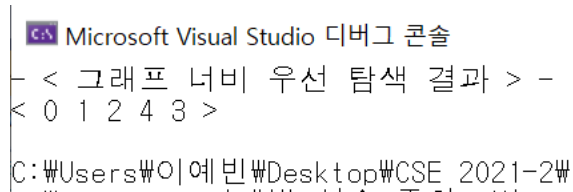
## 2.4 실행창

data.txt 파일은 다음과 같이 'v' 문자와 함께 정수형 정점이, 그리고 'e' 문자와 함께 두 정점이 저장되어 있다. 'v'인 경우는 정점을 추가, 'e'인 경우는 한 정점에서 다른 정점으로 향하는 간선을 삽입한다.



```
data.txt - Windows 메모장
파일(F) 편집(E) 서식(O) 보기(V) 도
v 0
v 1
v 2
v 3
v 4
e 3 4
e 2 4
e 2 3
e 1 2
e 0 4
e 0 2
e 0 1
```

너비 우선 탐색의 결과 실행창은 다음과 같이 나타남을 확인할 수 있다.



```
Microsoft Visual Studio 디버그 콘솔
- < 그래프 너비 우선 탐색 결과 > -
< 0 1 2 4 3 >
C:\Users\이예빈\Desktop\CSE_2021-2\...
```



## 2.5 느낀점

너비 우선 탐색은 큐 구조를 사용한다는 점에서 조금 복잡하지만, 순환적인 호출을 하는 깊이 우선 탐색에 비해서는 직관적이고 탐색의 순서가 헷갈리지 않은 것 같다는 느낌을 받았다. 문제를 풀면서, 2 번 문제의 파일의 데이터는 정점의 값이 0 부터 4 까지 중간에 비어있는 값이 없다는 점에서 행렬의 인덱스 번호를 그대로 정점의 값으로 사용해 볼까 하는 고민도 있었지만, 아무래도 코드를 재사용하고 이미 만들어놓았던 함수들을 그대로 활용하는 것이 훨씬 간단하다고 생각했기에 큰 어려움 없이 너비 우선 탐색을 하는 함수와 큐 구조와 관련된 몇 가지 연산들만 구현하여 2 번 문제를 잘 마무리 할 수 있었기에 뿌듯했다.

### 3. 느낀점

---

깊이 우선 탐색과 너비 우선 탐색을 이전에 그래프를 직접 그리고 탐색의 순서를 손으로 그려가며 따라가 보았을 때에는 어렵지 않았지만 코드로 구현하여 이를 이해하는 것이 쉽지만은 않았다. 하지만 배열의 인덱스 값을 이용해 그래프를 만드는 과정까지 추가하여 이번 8주차 문제들을 풀어보니, 깊이 우선 탐색과 너비 우선 탐색의 차이도 명확히 알 수 있었고 각 탐색의 코드들에 대한 이해도 많이 오른 것 같다. 인접 행렬과 리스트를 통해 그래프를 구현하는 과정도 더욱 명확하게 이해한 것 같고, 앞으로 그래프를 탐색하고 간선의 가중치를 이용하여 최단 거리 등을 구하는 알고리즘들도 열심히 배워나가보고 싶다.