

# 자료구조응용

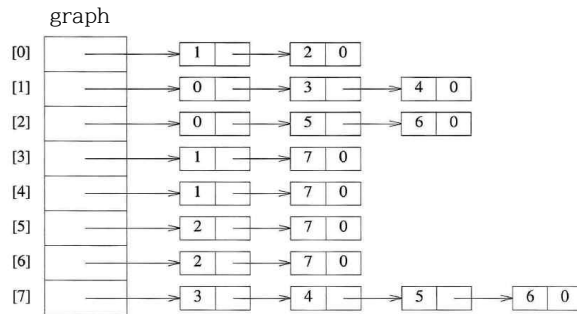
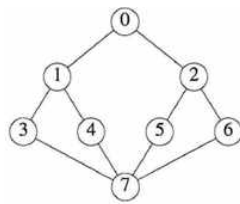
## 18. Graph: DFS, BFS (15점)

2022.5.11.

1. 다음과 같이 무방향그래프(undirected graph) 데이터를 입력받아 인접리스트를 만들고 dfs 결과를 출력하는 프로그램을 작성하라.(5점)

### (1) 입력파일(input.txt) 및 자료구조

```
8 10
0 1
0 2
1 3
1 4
2 5
2 6
3 7
4 7
5 7
6 7
```



- ※ 입력파일의 첫 줄은 정점(vertex) 수와 간선(edge)의 수를 나타냄
- ※ 그래프의 정점 인덱스는 0부터 시작됨
- ※ 주의: 파일로부터 구성된 인접리스트의 노드 순서가 그림(graph)과 동일하지는 않음

### (2) 실행순서

- ① 정점(vertex)과 간선(edge)의 수를 입력받음
- ② 그래프를 구성하는 간선을 하나씩 입력받으면서 인접리스트를 구성함
  - ※ 같은 간선이 두 번 입력되지 않음을 가정함
  - ※ 항상 헤더 다음인 처음 노드로 입력되게 함
- ③ dfs의 결과 출력
  - ※ Program 6.1의 재귀함수호출을 이용함. 시스템 스택의 사용
  - ※ dfs(0), dfs(1), ..., dfs(n)를 각각 출력함

### (3) 구현 세부사항

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

```
void dfs(int v)
{
    /* depth first search of a graph beginning at v */
    nodePointer w;
    visited[v] = TRUE;
    printf("%5d", v);
    for (w = graph[v]; w; w = w->link)
        if (!visited[w->vertex])
            dfs(w->vertex);
}
```

Program 6.1: Depth first search

#### (4) 실행 예

A screenshot of a Windows Command Prompt window titled "C:\Windows\system32\cmd.exe". The output shows two graphs being processed by Depth First Search (DFS).  
  
The first section displays an Adjacency List:  
Adjacency List  
graph[0] : 2 1  
graph[1] : 4 3 0  
graph[2] : 6 5 0  
graph[3] : 7 1  
graph[4] : 7 1  
graph[5] : 7 2  
graph[6] : 7 2  
graph[?] : 6 5 4 3  
  
The second section displays the results of DFS:  
Depth First Search  
dfs(0) : 0 2 6 7 5 4 1 3  
dfs(1) : 1 4 7 6 2 5 0 3  
dfs(2) : 2 6 7 5 4 1 3 0  
dfs(3) : 3 7 6 2 5 0 1 4  
dfs(4) : 4 7 6 2 5 0 1 3  
dfs(5) : 5 7 6 2 0 1 4 3  
dfs(6) : 6 7 5 2 0 1 4 3  
dfs(?) : 7 6 2 5 0 1 4 3  
  
At the bottom, there is Korean text: "계속하려면 아무 키나 누르십시오 . . ." which translates to "Press any key to continue...". A cursor is visible at the end of this message.

2. 위 1번 문제에 대해 dfs 대신 bfs의 결과를 출력하는 프로그램을 작성하라. (5점)

(1) 실행순서

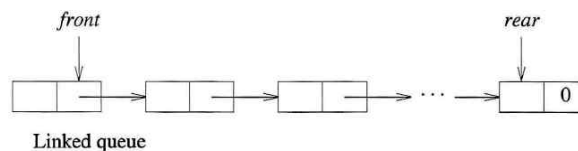
- ①, ② - 1번과 동일
- ③ bfs의 결과 출력
  - ※ Program 6.2 및 linked queue를 사용함
  - ※ bfs(0), bfs(1), ..., bfs(n)를 각각 출력함

(2) 구현 세부사항

### ① Linked Queue

The queue definition and the function prototypes used by *bfs* are:

```
typedef struct qNode *queuePointer;
typedef struct qNode {
    int vertex;
    queuePointer link;
} tQNode;
queuePointer front, rear;
void addq(int);
int deleteq();
```



---

```

void addq(int i, element item)
{
    /* add item to the rear of queue i */
    queuePointer temp;
    MALLOC(temp, sizeof(*temp));
    temp->data = item;
    temp->link = NULL;
    if (front[i])
        rear[i]->link = temp;
    else
        front[i] = temp;
    rear[i] = temp;
}

```

---

**Program 4.7:** Add to the rear of a linked queue

---

```

element deleteq(int i)
{
    /* delete an element from queue i */
    queuePointer temp = front[i];
    element item;
    if (!temp)
        return queueEmpty();
    item = temp->data;
    front[i] = temp->link;
    free(temp);
    return item;
}

```

---

**Program 4.8:** Delete from the front of a linked queue

※ Program 4.7~4.8은 다중 큐에 대한 함수이므로 단일 큐에 대한 함수로 수정 (즉, i와 관련된 부분을 삭제). element를 int로, data를 vertex로 수정

## ② bfs 함수

---

```

void bfs(int v)
{
    /* breadth first traversal of a graph, starting at v
       the global array visited is initialized to 0, the queue
       operations are similar to those described in
       Chapter 4, front and rear are global */
    nodePointer w;
    front = rear = NULL; /* initialize queue */
    printf("%5d", v);
    visited[v] = TRUE;
    addq(v);
    while (front) {
        v = deleteq();
        for (w = graph[v]; w; w = w->link)
            if (!visited[w->vertex]) {
                printf("%5d", w->vertex);
                addq(w->vertex);
                visited[w->vertex] = TRUE;
            }
    }
}

```

---

**Program 6.2:** Breadth first search of a graph

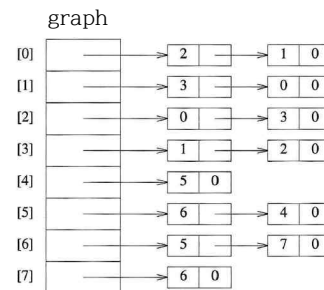
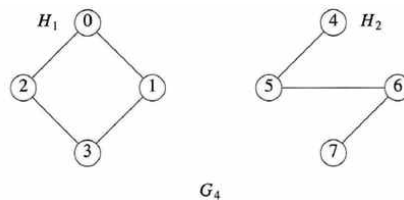
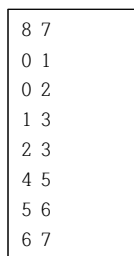
### (3) 실행 예

```
C:\Windows\system32\cmd.exe
```

```
<<<<<<<<< Adjacency List >>>>>>>>>  
graph[0] :    2     1  
graph[1] :    4     3     0  
graph[2] :    6     5     0  
graph[3] :    7     1  
graph[4] :    7     1  
graph[5] :    7     2  
graph[6] :    7     2  
graph[7] :    6     5     4     3  
  
<<<<<<<<< Breadth First Search >>>>>>>>>  
bfs(0) :   0  2  1  6  5  4  3  7  
bfs(1) :   1  4  3  0  7  2  6  5  
bfs(2) :   2  6  5  0  7  1  4  3  
bfs(3) :   3  7  1  6  5  4  0  2  
bfs(4) :   4  7  1  6  5  3  0  2  
bfs(5) :   5  7  2  6  4  3  0  1  
bfs(6) :   6  7  2  5  4  3  0  1  
bfs(7) :   7  6  5  4  3  2  1  0  
계속하려면 아무 키나 누르십시오 . . .
```

3. 입력된 무방향그래프의 connected component를 출력하는 프로그램을 작성하라. (5점)

### (1) 입력파일(input.txt) 및 자료구조



(2) 구현 세부사항

```
void connected(void)
{/* determine the connected components of a graph */
int i;
for (i = 0; i < n; i++)
    if(!visited[i]) {
        dfs(i);
        printf("\n");
    }
}
```

---

**Program 6.3: Connected components**

### (3) 실행 예

1번 그래프

G4

[illegible][illegible]

■ 제출 형식

- 솔루션 이름 : DS 18
- 프로젝트 이름 : 1, 2, 3
- 각 소스파일에 주석처리  
“학번 이름”

“본인은 이 소스파일을 다른 사람의 소스를 복사하지 않고 직접 작성하였습니다.”

- 실행화면을 캡처한 보고서를 작성 후 pdf 파일로 변환하여 솔루션 폴더에 포함
- 솔루션 정리 메뉴를 수행 후 전체 솔루션을 “학번.zip”으로 압축하여 제출

■ 주의

- 소스 복사로는 실력향상을 기대할 수 없습니다!!!
- 1차 마감: 수업일 자정
- 2차 마감: 수업 다음날 자정
- 위 마감 이외의 제출은 허용하지 않습니다. ( 이메일 제출 불가! )