**Computer Algorithm**

**Chapter 08**

**Graph**

**[실습] DFS, BFS 구현**

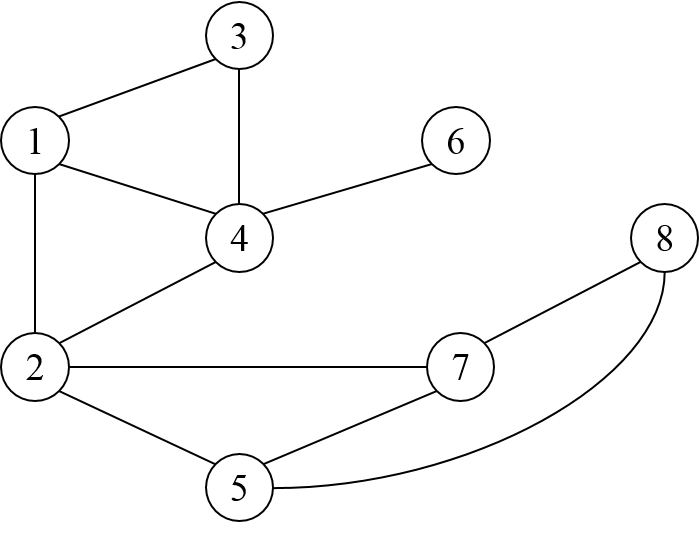
**실습 목표**

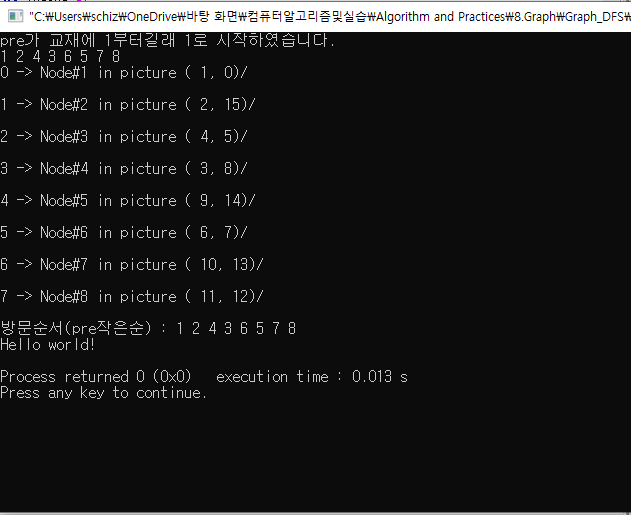
* **그래프를 배열로 구현하고 이를 문제해결에 활용한다.**
* **DFS, BFS가 어떤 방식으로 작동하는지 이해한다.**
* **DFS, BFS가 어떠한 차이가 있고 어떠한 경우에 사용하는 지에 대해 이해할 수 있다.**
* **다른 완전탐색방식에는 어떤 것이 있는지 알아보고 학습한다.**

**요구사항**

* **실습과제 1) (30점)**

아래에 주어진 그래프를 배열로 구현하고 구현한 그래프를 DFS를 수행하는 알고리즘을 구현한다. 단 재귀함수을 사용 한다. 시작노드는 1이며 각 노드별 방문순서를 출력하도록 한다.

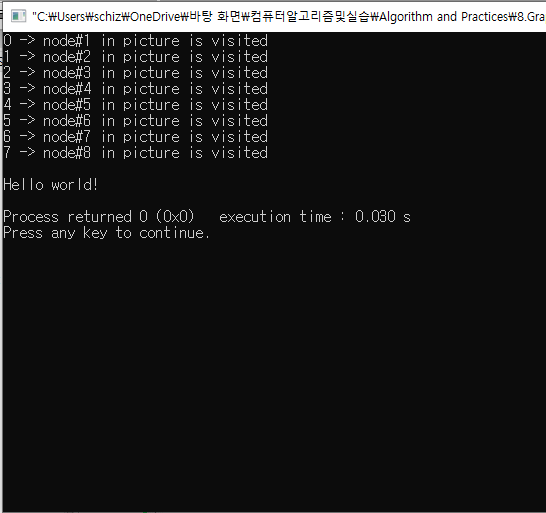
****



* **실습과제 2) (30점).**

실습과제 1에 있는 그래프로 BFS를 수행하는 알고리즘을 구현한다.

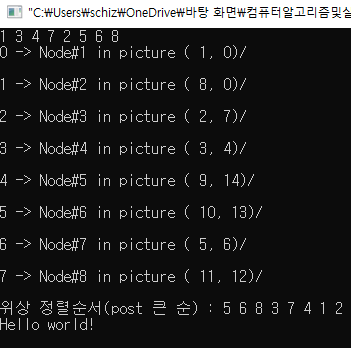
\* 입력은 실습과제 1과 같으며 출력은 노드들의 방문순서를 출력한 다.



* **실습과제 3) (40점).**

단방향그래프에 대한 배열을 작성하고 위상정렬을 구현한다.

\* 출력은 위상정렬된 노드 번호를 정렬된 순서대로 출력한다.



**배경지식**

그래프는 다양한 컴퓨터 분야의 문제해결에 사용되는 방법으로 현실 세계의 다양한 문제들을 객체와 객체간의 관계로 추상화하는 데 사용된다. 그래프로 추상화된 현실세계는 그래프 탐색 알고리즘을 통해 문제를 해결 할 수 있도록 하는데, 이러한 그래프 탐색은 인공지능 분야에서 매우 광범위하게 사용되는 방법으로 크게 깊이우선 탐색(DFS)과 너비우선 탐색(BFS)으로 나누어진다.

본 실습에서는 그래프를 배열로 구현하고 이를 통해 깊이우선 탐색과 너비우선 탐색을 구현하도록 한다.

1)

깊이 우선 탐색(Depth First Search)은 아래와 같은 과정으로 진행된다.

1. 시작점을 출발, 작은 번호 순서로 이동할 위치를 조사하고, 이동할 수 있는 정점까지 이동한다.
2. 이동할 곳이 없어지면 이동할 곳이 남아 있는 정점까지 돌아간 후, 다시 이동할 수 있는 정점까지 진행한다.
3. 더 이상 이동할 정점이 남이 있지 않으면 종료한다.

Depth First Search는 일반적으로 재귀를 통해 구현되며 이에 대한 pseudo code는 다음과 같다.

DFS(v)

{

visited[v] = true; // 노드의 방문순서 기록

pre[v] = i++;

for all vertices u adjacent to v // G[v][u] == 1 인 경우 인접해있다.

if visited[u] == 0 // 처음 방문한 노드일 경우

DFS(u);

post[u]= i++;

}

depthFirstSearch()

{

for all vertices v

num[v] = 0;

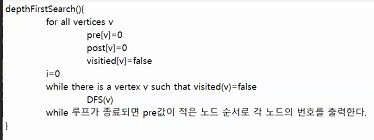
i = 0;

while there is a vertex v such that num(v) is 0

DFS(v)

pre값이 적은 노드 순서로 각 노드의 번호를 출력한다.

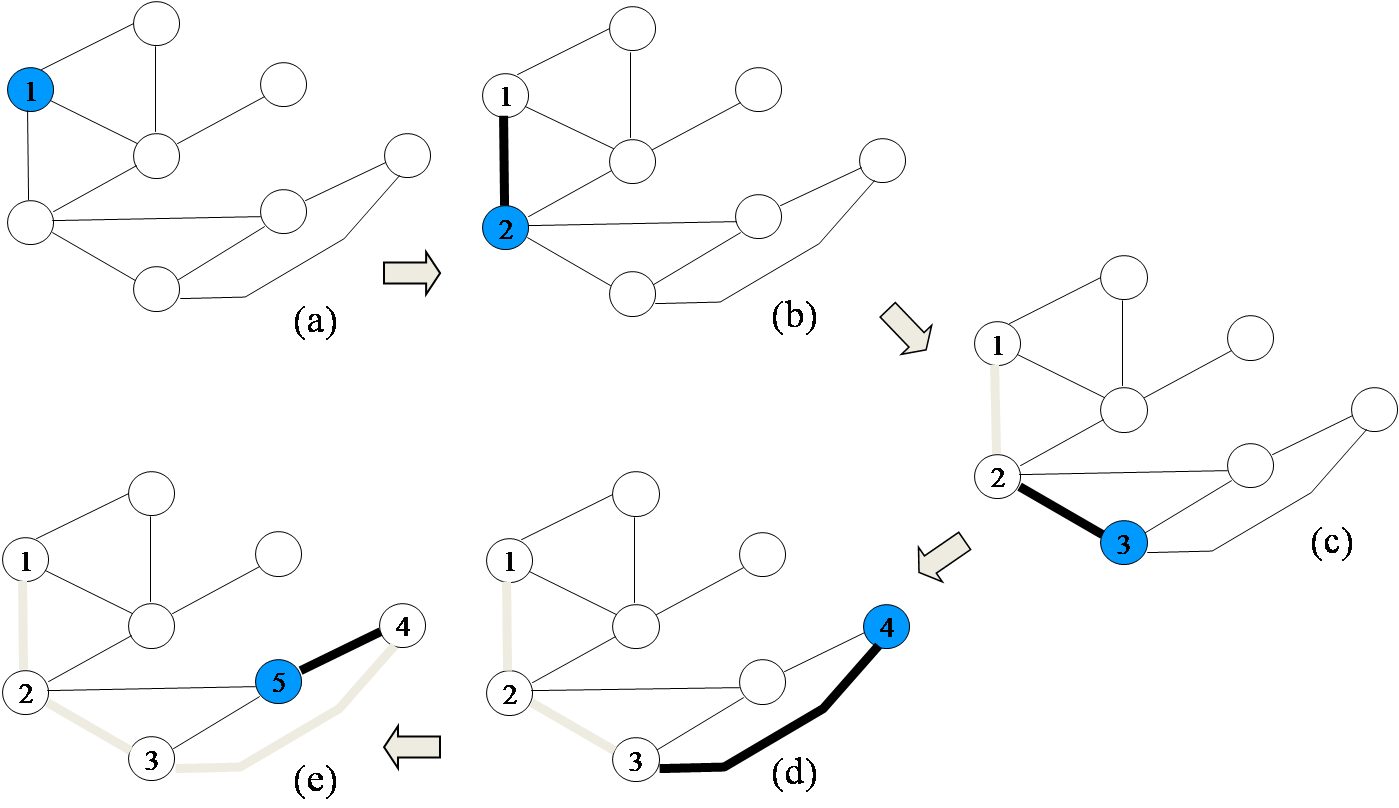
}

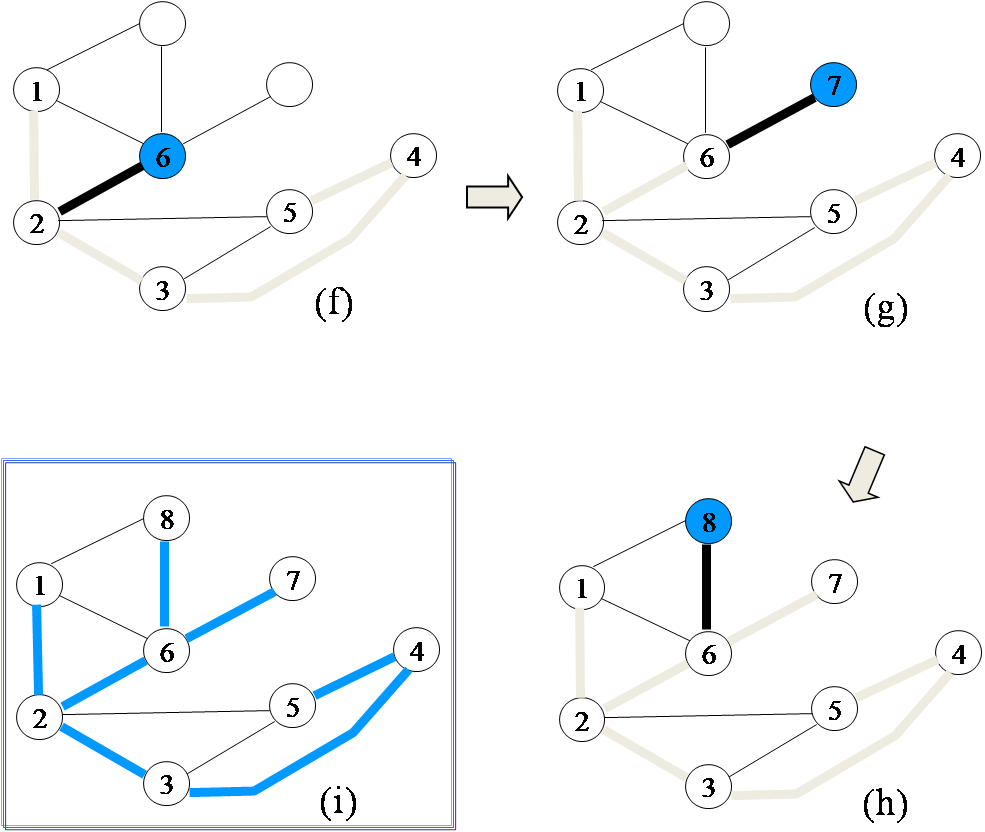


G[][]는 그래프를 표현하는 배열로 그래프 G의 두 노드 u,v사이에 edge가 있으면 G[u][v] = 1, 아니면 G[u][v] = 0으로 표현된다. visited[]은 노드들의 방문여부를 나타내는 배열로 0으로 초기화되어 있어야 한다. 따라서 visited[v]의 값이 0이면 노드 v는 방문이 안 되었으며 0이 아닌 경우 이미 방문을 했던 노드임을 나타낸다.

for all vertices u adjacent to v 구문의 u와 v의 인접여부는 그래프 G[u][v]의 값을 통해 확인할 수 있으며, if visited[u] == 0 방문되지 않은 노드에 대해서만 작업을 수행한다는 것을 의미한다.

다음은 Depth First Search의 작동 예이다.





2)

BFS를 구현하기 위해서는 queue가 필요하며 따라서 queue 자료구조와 함께 enqueue(), dequeue(), isEmpty(), isFull() 함수를 구현하여야 한다. 아래는 queue를 배열로 구현한 예이며 int type을 가지는 것으로 가정한다.

**int** Queue[QSize];

**int** first = last = 0;

**int** isFull() {

// first는 0, last는 QSize-1이거나 first가 last보다 큰 경우 큐가 full

**return** ((first == 0 && last == QSize – 1) || (first == last + 1))

}

**int** isEmpty() { // first와 last가 같은 경우 큐는 empty

return first == last;

}

**int** enqueue(int v) {

**if**(!isFull()) {

last = (last+1)% QSize;

Queue[last] = v;

}

**else** return -1; // 큐가 다 찼다면 return fail

**return** v; // return success

}

**int** dequeue() {

**int** v;

**if**(!isEmpty()) {

v = Queue[first];

first = (first+1)% QSize;

}

}

**else** return -1; // 큐가 비어 있다면 return fail

**return** v; // 해당 원소를 리턴

}

BFS 알고리즘을 표현한 슈도코드이다. 아래 슈도코드를 참고하여 구현한다.

BFS()

For all vertices u // num 초기화

visited[u] = 0;

head = tail = null;

i = 1;

while there is a vertex v such that num[v] == 0

// pick any v whose

enqueue(v);

while(queue is not empty)

w = dequeue();

print w // w의 id를 출력한다.

for all vertices u adjacent to w

// w와 인접한 노드 u는 인접행렬을 통해 찾는데,

// G[w][u] == 1 이면 u는 인접해 있다.

if(visited[u] == 0)

// 이미 방문한 노드가 아닐 경우

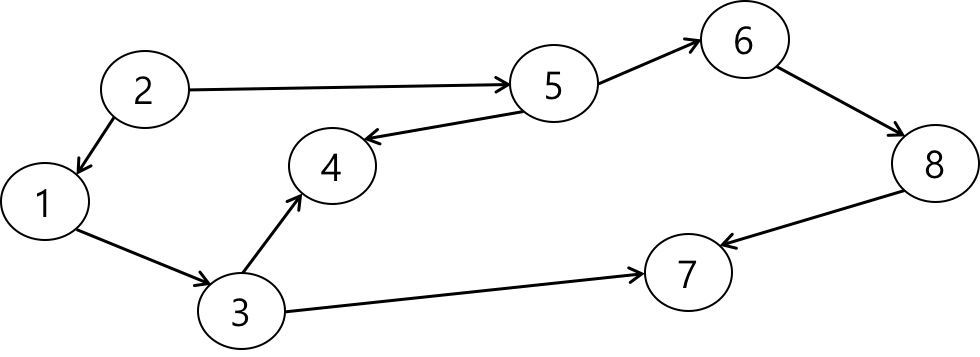
visited[u] = i++;

// 노드의 방문순서 기록

enqueue(u);

3)

위상정렬에 대한 이해를 위해 아래의 그래프를 참고해 보자. 아래의 그래프는 작업의 순서를 그래프로 나타낸 것이다. 이 그래프에는 부분 순서 관계가 있다. 1번 작업을 하기 위해서는 2번 작업을 먼저 완수해야 한다. 7번 작업을 진행하기 위해서는 3번, 또는 6번 또는 8번 작업을 완료해야 한다.



이때 작업의 순서는 2, 1, 3, 4, 7과 2, 5, 4, 6, 7과 같은 두 가지 계열로 나누어 있음을 알 수 있다. 1번 작업과 5번 작업은 계열이 다르므로 어떤 작업을 먼저 수행해도 상관없다. 이와 같이 전체 그래프의 모든 노드가 아닌 일부 노드에 대해서 선후 관계를 가지는 경우 이를 부분 순서 관계(partial order)라고 한다.

이러한 부분 순서 관계에 대해 먼저 진행할 작업부터 나중에 진행할 작업까지 일렬로 나열하는 것을 topological sort(위상정렬)이라고 한다. 교재에서는 linearization이라는 용어로 소개되어 있다.

위상 정렬을 위해서는 먼저 그래프가 directed graph(방향성 그래프)이어야 하며 depth first search(깊이 우선 탐색)을 수행하고 각 정점에 pre와 post 숫자를 할당한다. (실습예제 1의 DFS참고) 이 후 할당된 post 숫자에 따라 post 숫자가 높은 정점부터 정렬시키면 된다.

예를 들면 위 그래프에서 깊이우선 탐색을 한 결과 pre,post 숫자가 다음 표와 같이 구해진다면 topological sort의 결과는 아래와 같이 나타난다.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 노드번호 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| pre | 1 | 9 | 2 | 3 | 10 | 11 | 5 | 12 |
| post | 8 | 16 | 7 | 4 | 15 | 14 | 6 | 13 |

2 5 6 8 1 3 7 4

위 표를 보면 post 값이 가장 높은 노드는 노드 2 (post 값 16) 이며, 가장 post 값이 가장 낮은 노드는 노드 4 (post 값 4) 이다. 따라서 위상 정렬의 결과 노드 2가 가장 높은 우선 순위를 가지는 노드가 되며, 노드 4가 가장 낮은 우선 순위를 가지는 노드가 된다.

위상 정렬은 부분 순서관계에 따라 정렬하는 것이기 때문에, 깊이우선 탐색의 시작 정점 및 깊이 우선 탐색의 탐색 방식에 따라 다양한 부분 순서를 가질 수 있다.

**제출방법**

* 보고서 작성방법: 실습문제 번호별로 결과가 나온 화면의 내용을

캡쳐하여 보고서에 붙여 놓는다.

* 소스코드의 파일이름에 연습문제 번호를 붙이는 것을 잊지 않는다. 예) ex-1.c, ex-2.c
* 결과 보고서에 이름과 작성 날짜를 기입하는 것을 잊지 않는다. 예) 김웅섭\_2020\_09\_01.doc
* 실행결과를 보고서에 작성하여 소스코드와 함께 제출한다.
* 제출 마감 : e-class 제출 마감시간까지