Data Structure #12

Graph

2020년 1학기

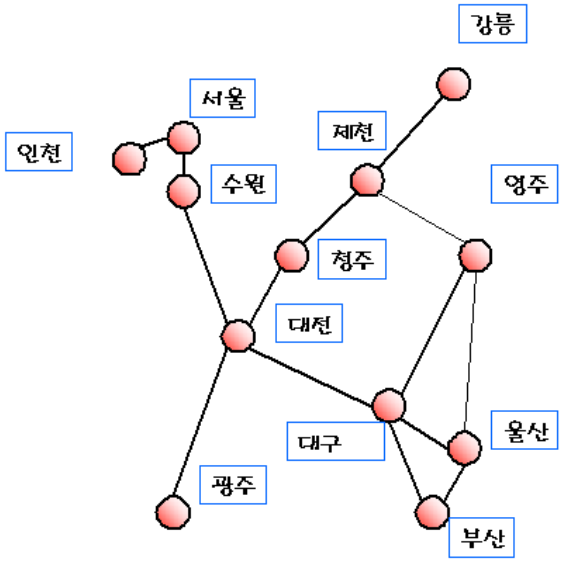
Intro.

- **실습주제 소개**
  - Graph DFS/BFS, topological sort
- **실습수업 문제**
  - Graph DFS
  - Graph BFS

CSLAB

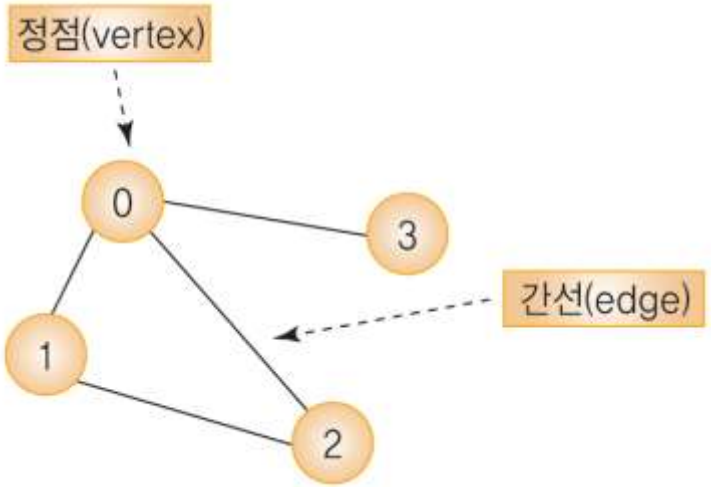
그래프(Graph)

- 연결되어 있는 객체 간의 관계를 표현하는 자료구조
- 가장 일반적인 자료구조 형태
  - 우리가 배운 트리(tree)도 그래프의 특수한 경우임
  - 지도에서 도시들의 연결 상태



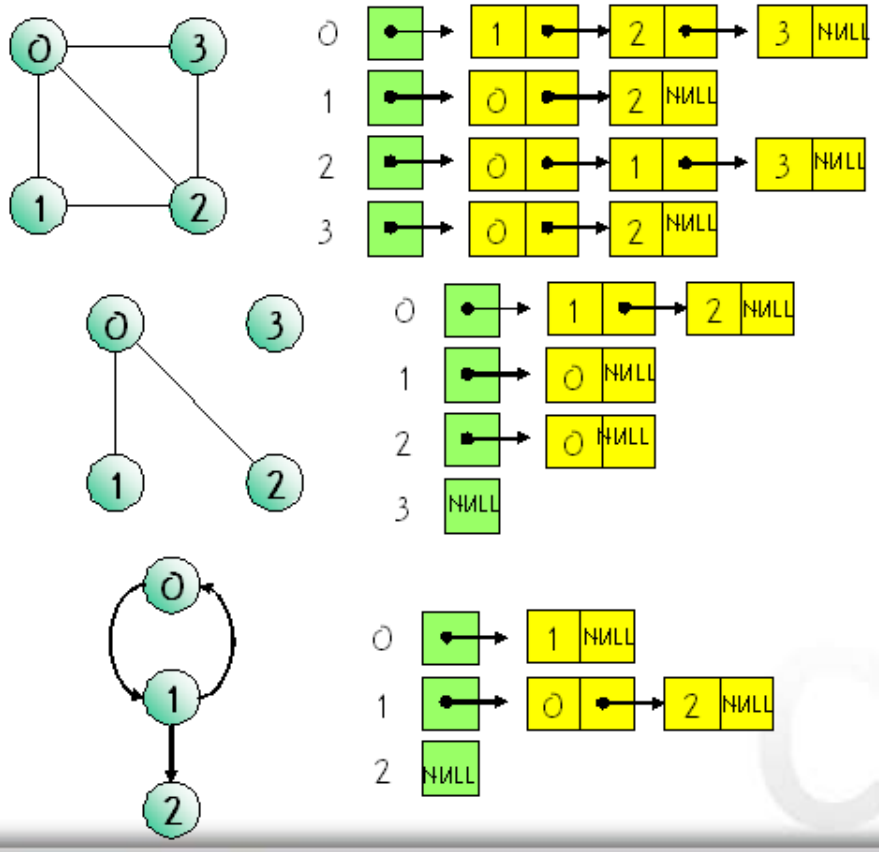
그래프(Graph) 정의

- **그래프 G는 (V, E)로 표시**
  - 정점(vertices)
    - 여러 가지 특성을 가질 수 있는 객체
    - V(G) : 그래프 G의 정점들의 집합
    - 노드(node)
  - 간선(edge)
    - 정점들 간의 관계
    - E(G) : 그래프 G의 간선들의 집합
    - 링크(link)



그래프 표현 방법(cont.)

- **인접리스트 (adjacency list)**
  - 각 정점에 인접한 정점들을 연결리스트로 표현

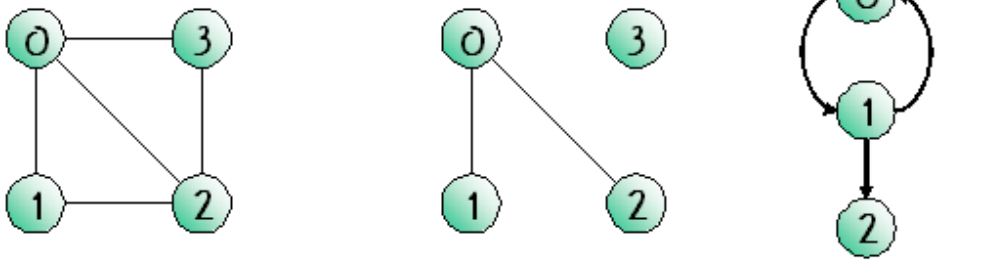


그래프 표현 방법

- **인접행렬 (adjacent matrix)**

  if(간선 (i, j)가 그래프에 존재) M[i][j] = 1,
  else M[i][j] = 0.

- **무방향 그래프의 인접 행렬은 대칭**



	0	1	2	3
0	0	1	1	1
1	1	0	1	0
2	1	1	0	1
3	1	0	1	0

(a)

	0	1	2	3
0	0	1	1	0
1	1	0	0	0
2	1	0	0	0
3	0	0	0	0

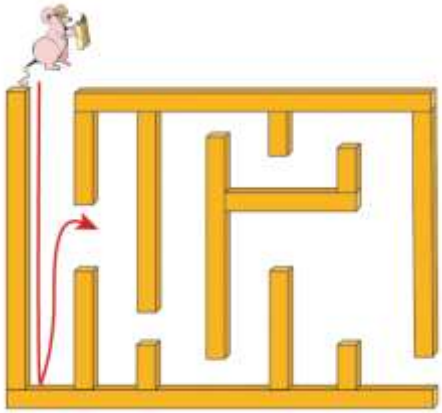
(b)

	0	1	2
0	0	1	0
1	1	0	1
2	0	0	0

(c)

깊이 우선 탐색(DFS)

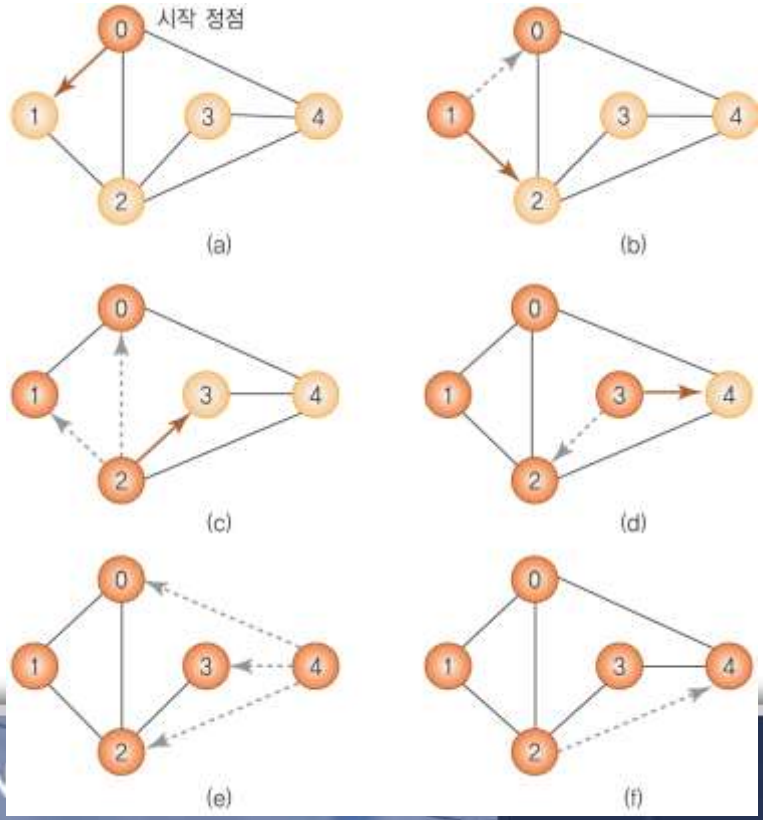
- **깊이 우선 탐색 (DFS: depth-first search)**
  - 루트 노드(혹은 다른 임의의 노드)에서 시작해서 다음 분기(branch)로 넘어가기 전에 해당 분기를 완벽하게 탐색하는 방법
  - 한 방향으로 갈 수 있을 때까지 진행, 더 이상 갈 수 없게 되면 가장 가까운 갈림길로 돌아와서 이 곳으로부터 다른 방향으로 다시 탐색 진행
  - 되돌아가기 위해서는 스택 필요(그러나 연결된 노드 중 방문된 적이 없는 노드를 재귀함수로 호출하면 스택이 필요 없음)



깊이 우선 탐색(DFS)

- **알고리즘**

```
depth_first_search(v)
        v를 방문되었다고 표시;
        for all u ∈ (v에 인접한 정점) do
          if (u가 아직 방문되지 않았으면) then depth_first_search(u)
```



그래프(graph) 구현 DFS (개인 실습 #1)

- 구조체

```
#include <stdio.h>
#define MAX_VERTICES 4

typedef struct graph
{   int node;
    struct graph *link;
} list;
list *adj_list[MAX_VERTICES];

int adj_mat[MAX_VERTICES][MAX_VERTICES]={
{0,1,0,1},
{1,0,1,1},
{0,1,0,1},
{1,1,1,0} };
int visited[MAX_VERTICES];
int n=4;
```

그래프(graph) 구현 DFS

- DFS함수

```
void graph_dfs_mat(int v)
{
   int w;
/* fill in the blank */
}
```

pseudocode:
depth_first_search(v)
v를 방문되었다고 표시;
for all w ∈ (v에 인접한 정점) do
if (!visited[w]) then depth_first_search(w)

그래프(graph) 구현 DFS

- 메인함수

```
void main()
{
        graph_dfs_mat(0);
}
```

출력결과 ex)

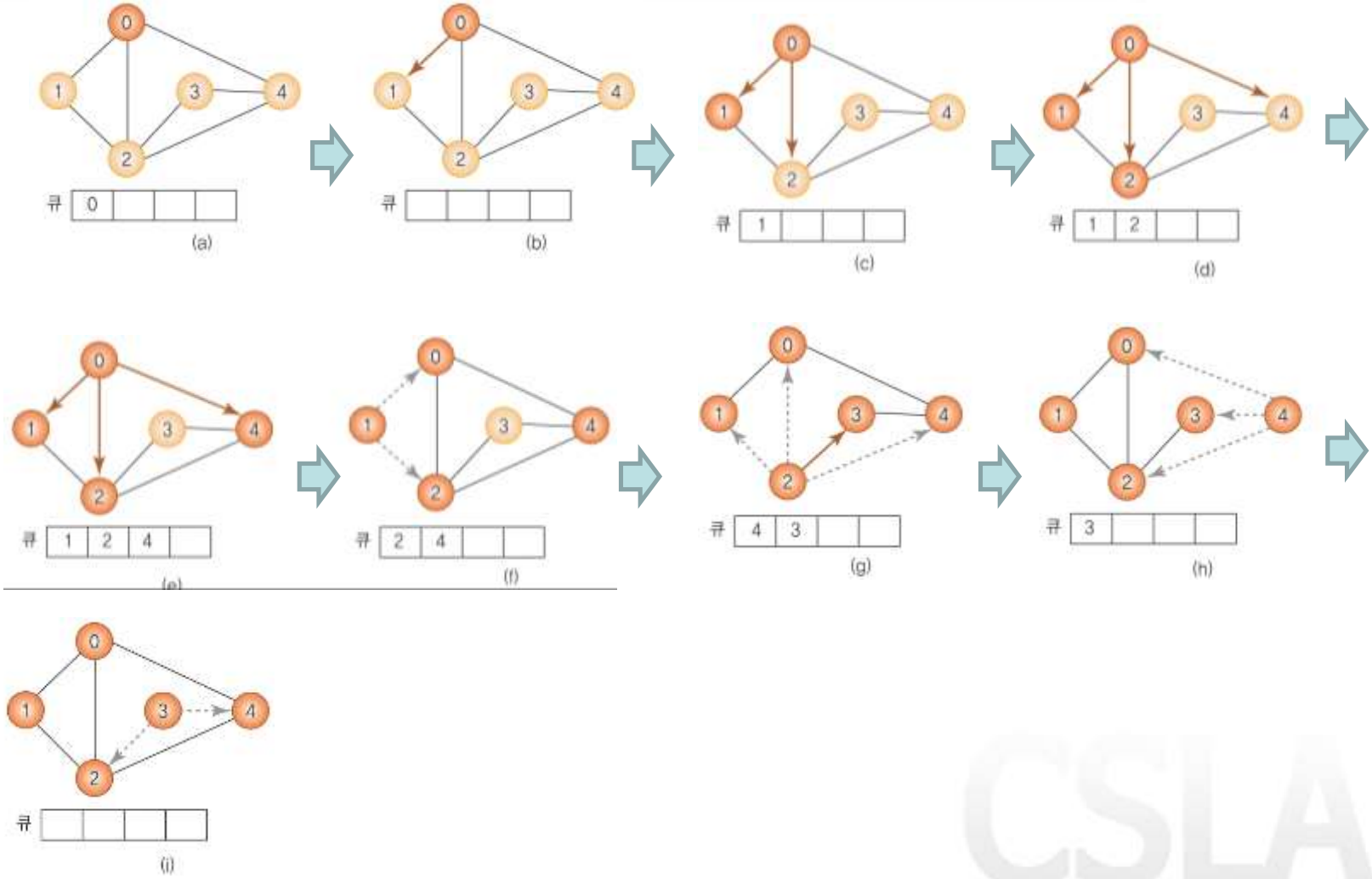
```
0 1 2 3
```

너비 우선 탐색(BFS)

- **너비 우선 탐색(BFS: breadth-first search)**
  - 루트 노드(혹은 다른 임의의 노드)에서 시작해서 인접한 노드를 먼저 탐색하는 방법
  - 시작 정점으로부터 가까운 정점을 먼저 방문하고 멀리 떨어져 있는 정점을 나중에 방문하는 순회 방법
  - 큐를 사용하여 구현됨
- **너비우선탐색 알고리즘**

```
breadth_first_search(v)
v를 방문되었다고 표시;
큐 Q에 정점 v를 삽입;
while (not is_empty(Q)) do
        Q에서 정점 w를 삭제;
        for all u ∈ (w에 인접한 정점) do
                if (u가 아직 방문되지 않았으면) then    u를 큐에 삽입;
                                                        u를 방문되었다고 표시;
```

너비 우선 탐색(BFS)



그래프(graph) 구현 BFS (개인 실습 #2)

- 구조체

※
코드 추가

```
#include <stdio.h>
#define MAX_VERTICES 4
#define MAX_QUEUE_SIZE 10

typedef struct graph
{  int node;
   struct graph *link;
} list;
list *adj_list[MAX_VERTICES];

int adj_mat[MAX_VERTICES][MAX_VERTICES]={
{0,1,0,1},
{1,0,1,1},
{0,1,0,1},
{1,1,1,0} };
int visited[MAX_VERTICES];
int n=4;

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

그래프(graph) 구현 BFS

– 큐 관련 함수

```
// 초기화 함수
void init(QueueType *q)

// 공백 상태 검출 함수
int is_empty(QueueType *q)

// 포화 상태 검출 함수
int is_full(QueueType *q)

// 삽입 함수
void enqueue(QueueType *q, element item)

// 삭제 함수
element dequeue(QueueType *q)
```

그래프(graph) 구현 BFS

- BFS함수

```
void graph_bfs_mat(int v)
{
        int w;
        QueueType q;
        init(&q);     /* 큐 초기화 */
        visited[v] = 1;   // 정점 v 방문 표시
        printf("%d ", v);

        ……

}
```

```
breadth_first_search(v)
v를 방문되었다고 표시;
큐 Q에 정점 v를 삽입;
while (not is_empty(Q)) do
        v = Q에서 정점 w를 삭제(aka. dequeue(Q));
        for all u ∈ (w에 인접한 정점) do
                if (w가 아직 방문되지 않았으면) then
                    w를 방문되었다고 표시;
                    w를 큐에 삽입;
```

그래프(graph) 구현 BFS

- 메인함수

```
void main()
{
        graph_bfs_mat(0);
}
```

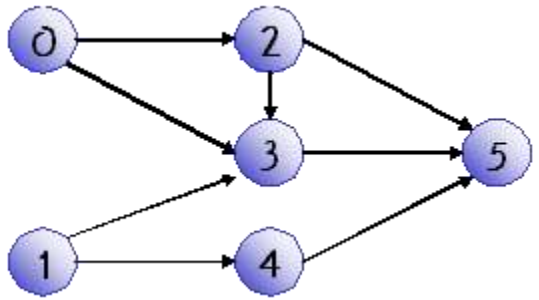
출력결과 ex)

```
0 1 3 2
```

위상정렬(topological sort)

- **위상정렬(topological sort):**
  - 방향 그래프에서 간선 <u, v>가 있다면 정점 u는 정점 v를 선행한다고 말한다. 방향 그래프에 존재하는 각 정점들의 선행 순서를 위배하지 않으면서 모든 정점을 나열하는 것을 방향 그래프의 위상 정렬(topological sort)이라고 한다.

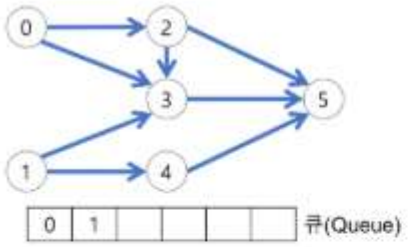
과목번호	과목명	선수과목
0	전산학개론	없음
1	이산수학	없음
2	자료구조	1
3	알고리즘 분석	0, 1, 2
4	운영체제	1
5	인공지능	2, 3, 4



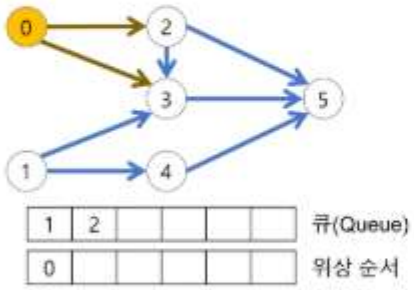
  - 위상정렬: (0,1,2,3,4,5) , (1,0,2,3,4,5)
  - (2,0,1,3,4,5)는 위상 정렬이 아니다. 왜냐하면 2번 정점이 0번 정점 앞에 오기 때문이다. 간선 <0, 2>이 존재하기 때문에 0번 정점이 끝나야 만이 2번 정점을 시작할 수 있다.

위상정렬(topological sort)

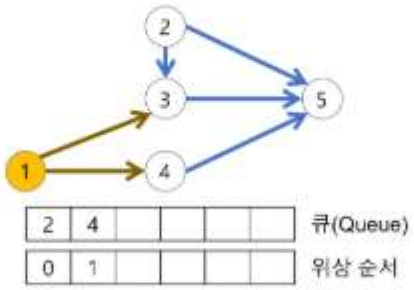
(1) 초기 상태



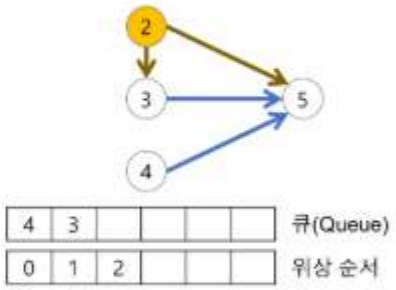
(2) 0 제거



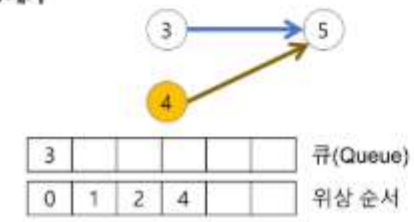
(3) 1 제거



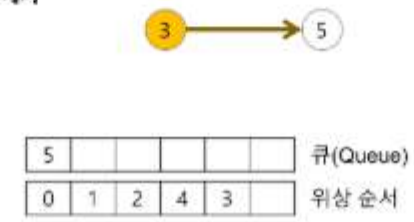
(4) 2 제거



(5) 4 제거



(6) 3 제거



(7) 5 제거

제출

- **제출**
  - 개인 실습 (#1, #2)
    - 오늘 자정까지 제출 (~ 2020/6/5 23:59)
    - DFS, BFS
  - 과제
    - Lab12.docx
    - 다음주 목요일 자정까지 제출 (~2020/6/11 23:59)
    - Topological sort

과제

- **Lab12.docx**

Input.txt

```
1 2 3 4 5 6
1-2 1-4 2-5 2-4- 2-3 3-4 5-3 6-3 6-5
```

output

1 6 2 5 3 4