Data Structure

Week 14

KyuDong SIM

Data Structure

Week 13

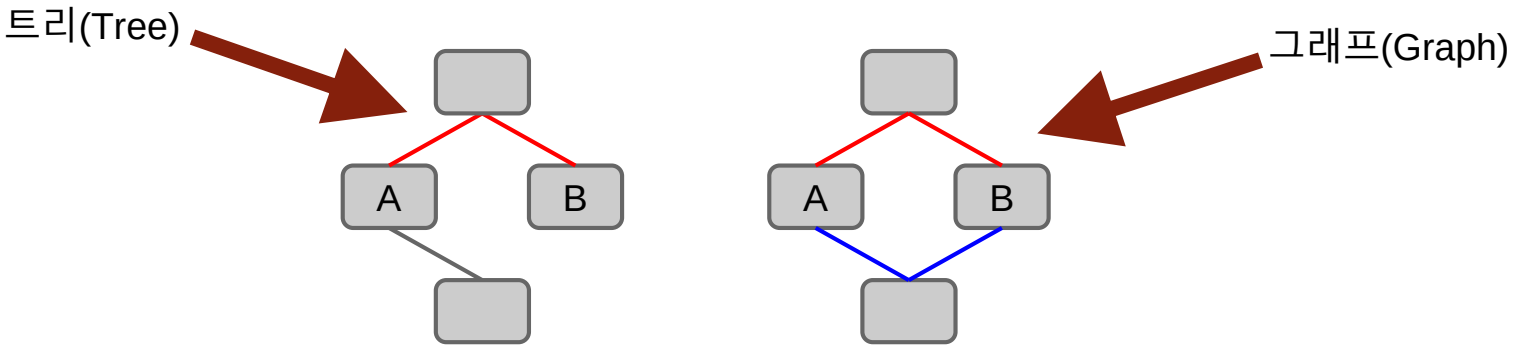
KyuDong SIM

1. 이번 주 실습 내용

- Graph의 DFS 와 BFS 구현

Graph란?

- A 노드에서 B 노드까지 경로가 유일하지 않을 수 있음
- Tree는 Graph의 특수한 경우

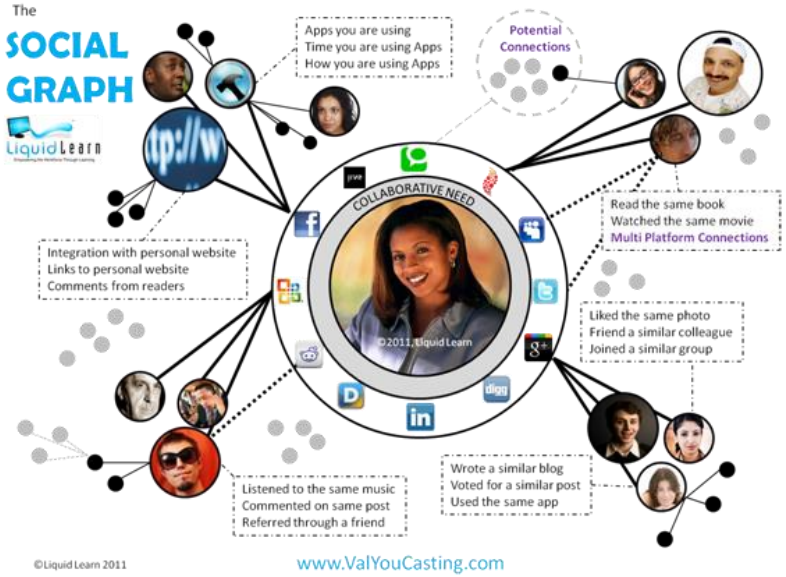


Graph의 요소와 종류

- 그래프 요소
  - Vertex (Node)
  - Edge (Link)
- 무향 그래프 (Undirected Graph)
  - 방향성이 없음
- 유향 그래프 (Directed Graph)
  - 방향성이 있음
- 가중치 그래프 (Weighted Graph)
  - 각 Edge에 가중치가 부여됨

Graphs 가 적용된 사례

- Social Graph(Social Network)

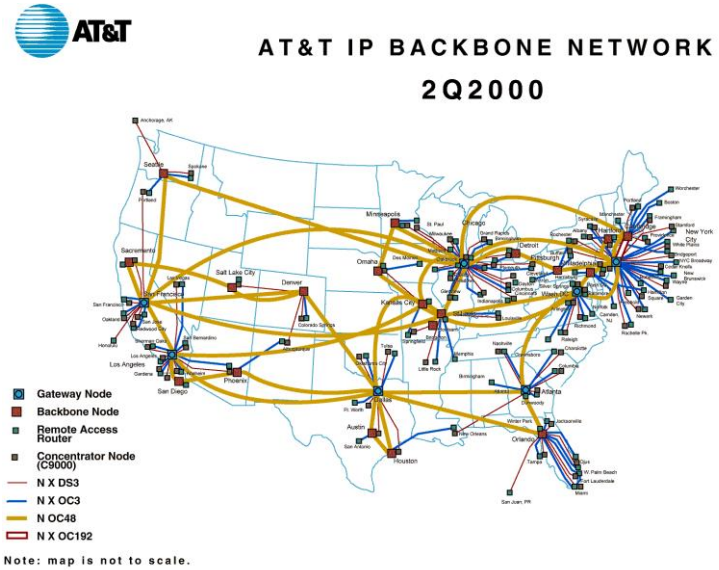


Graphs 가 적용된 사례

- Automotive Navigation System

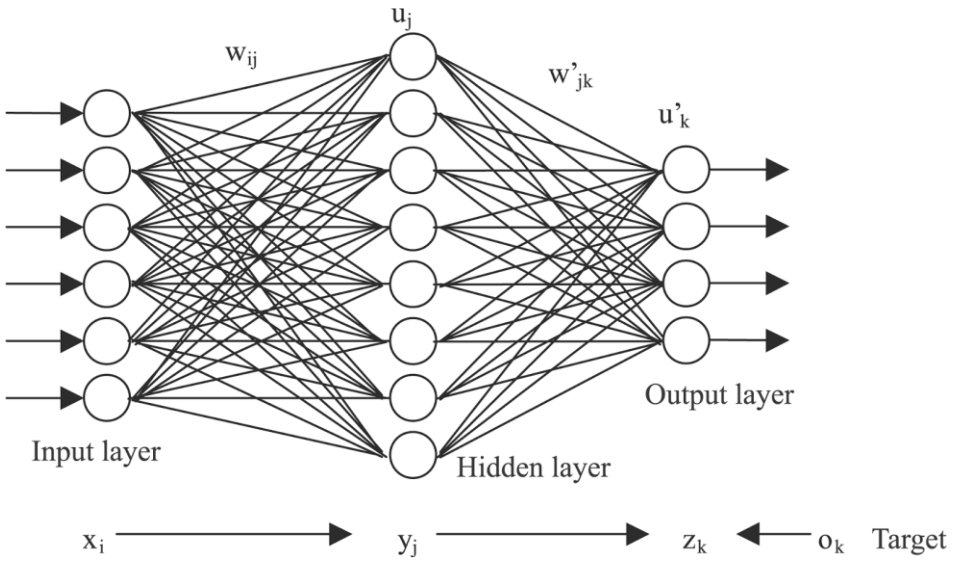
Graphs 가 적용된 사례

- Internet Network



Graphs 가 적용된 사례

- Artificial Neural Network

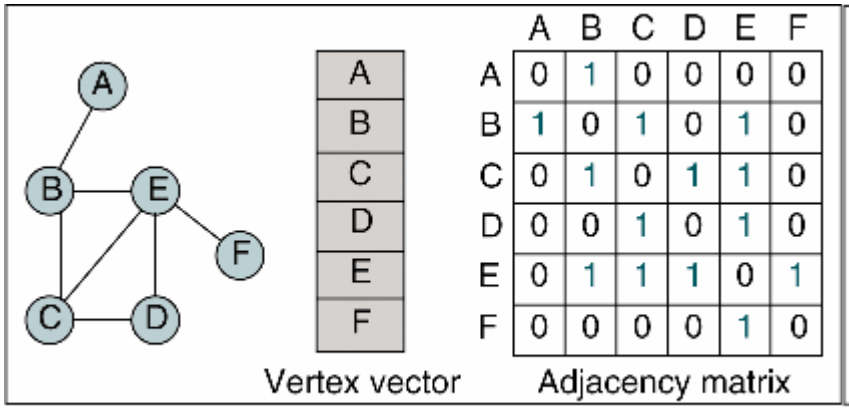


Graphs의 표현방법

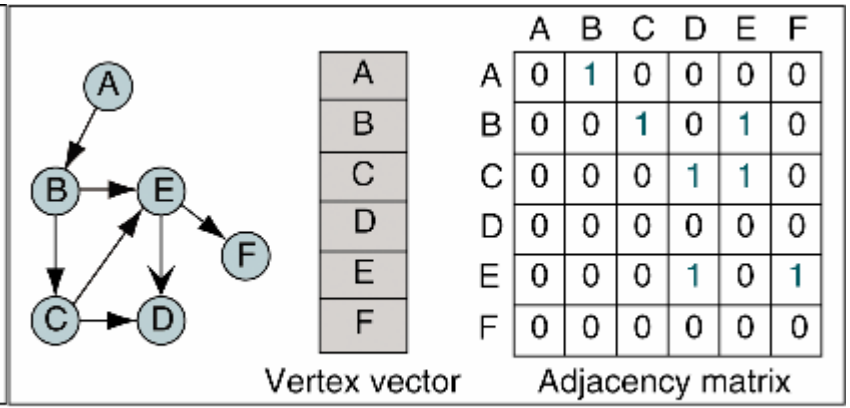
- Adjacency Matrix
- Adjacency List
- Sparse Matrix Representation
- Adjacency Multilists

Graphs의 표현방법

- Adjacency Matrix



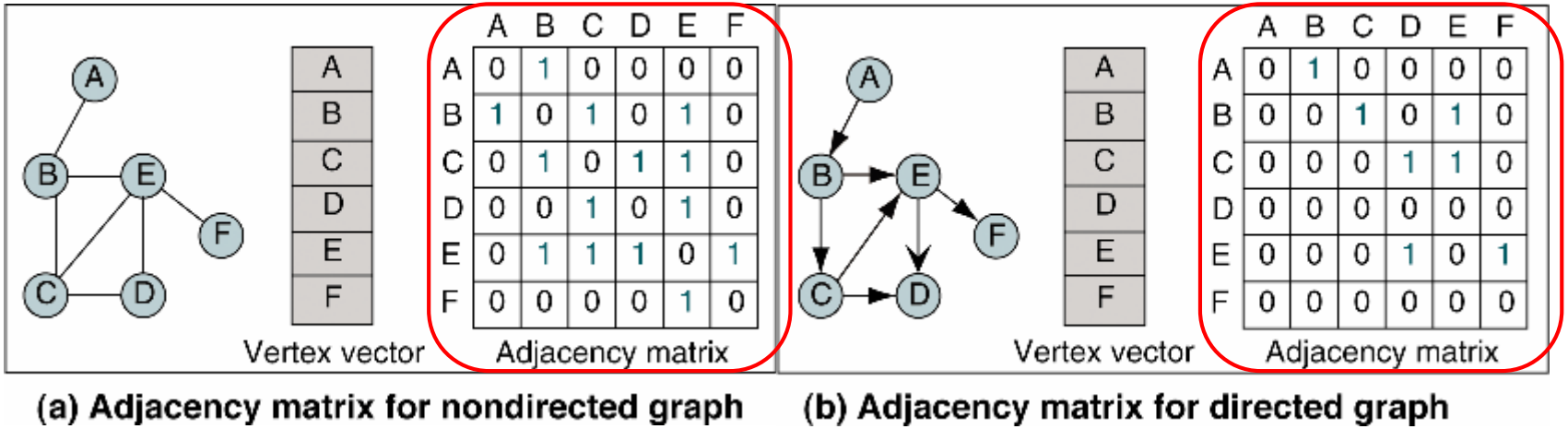
(a) Adjacency matrix for nondirected graph



(b) Adjacency matrix for directed graph

Graphs의 표현방법

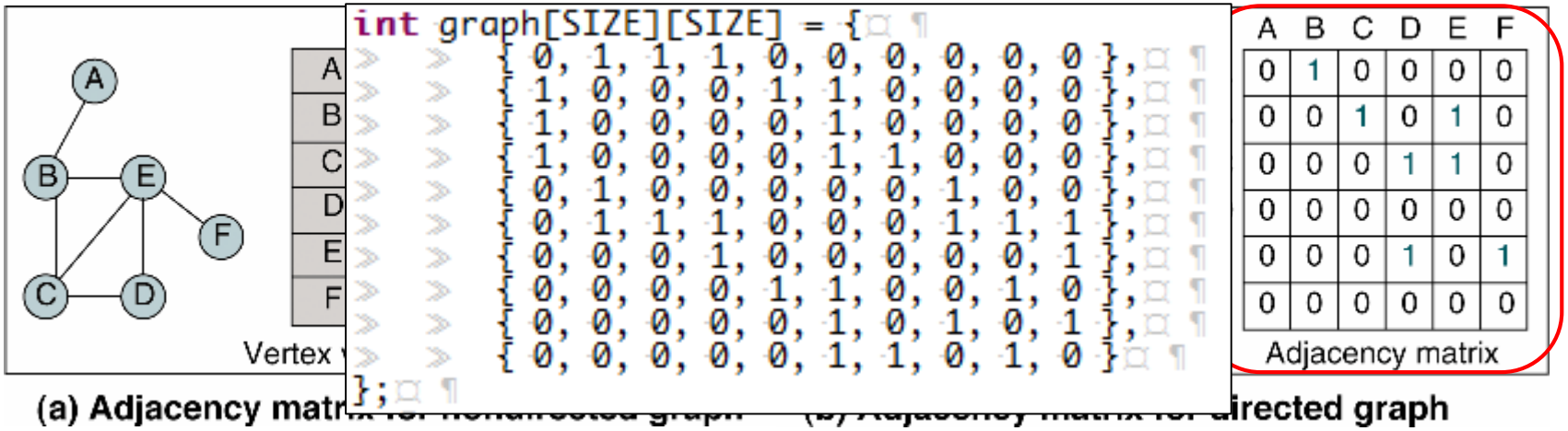
- Adjacency Matrix



(a) Adjacency matrix for nondirected graph **(b) Adjacency matrix for directed graph**

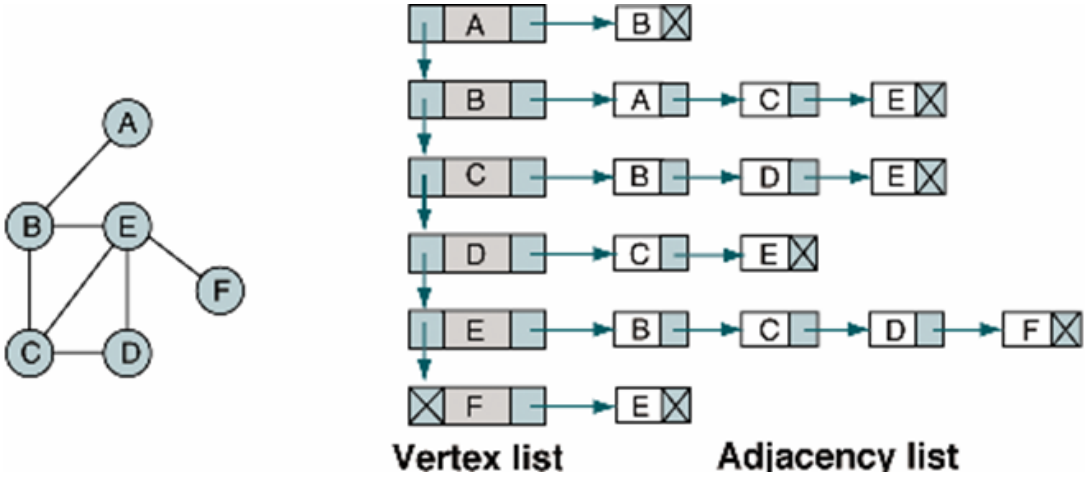
Graphs의 표현방법

- Adjacency Matrix



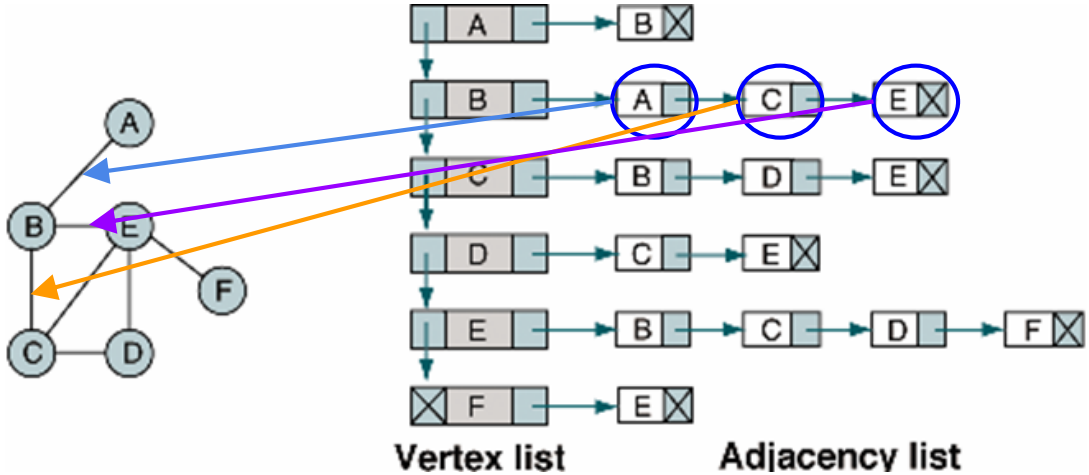
Graphs의 표현방법

- Adjacency List
- 링크(포인터)를 이용하여 리스트 형태로 표현



Graphs의 표현방법

- Adjacency List
- 링크(포인터)를 이용하여 리스트 형태로 표현



Graphs의 표현방법 (More)

- Sparse Matrix Representation

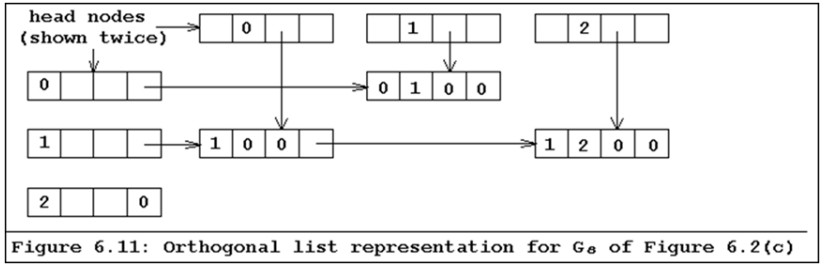


Figure 6.11: Orthogonal list representation for G_8 of Figure 6.2(c)

- Adjacency Multilists

m	vertex1	vertex2	list1	list2

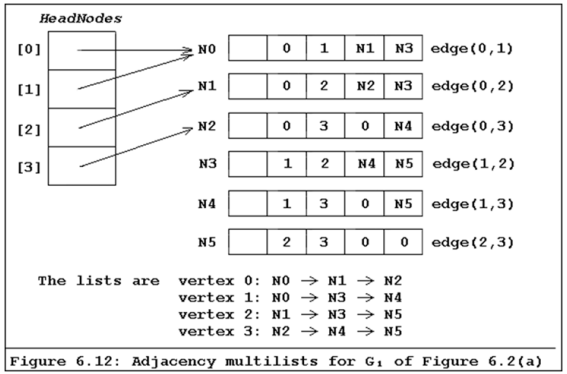


Figure 6.12: Adjacency multilists for G_1 of Figure 6.2(a)

Graph Searching

- Tree는 그래프의 특수한 경우!!
- Depth First Search (DFS)
  - 깊이 우선 탐색
- Breath First Search (BFS)
  - 너비 우선 탐색

Tree Traversal

- Stack 기반 (참고 : Depth First Search)
  - Pre-order : 전위 탐색
  - In-order : 중위 탐색, 대칭 탐색
  - Post-order : 후위 탐색
  - Recursive 호출을 통해 구현 가능 (함수 호출이 스택과 동일한 효과임을 이용)
- Queue 기반 (참고 : Breath First Search)
  - Level-order
  - Queue 자료 구조로 구현

Depth First Search

- Depth First Search
  - Stack을 기반으로 사용
  - 방문한 정점에 인접하면서 아직 방문하지 않은 정점을 선택하여 탐색
  - 방문하지 않은 정점이 더 이상 없다면, 스택에서 정점을 하나씩 꺼내서 위의 과정 반복
  - 스택이 비워지면 종료
- Time Complexity
  - O(|V| + |E|) : 정점의 개수 + 링크의 수
- Space Complexity
  - O(|V|) : 정점의 개수

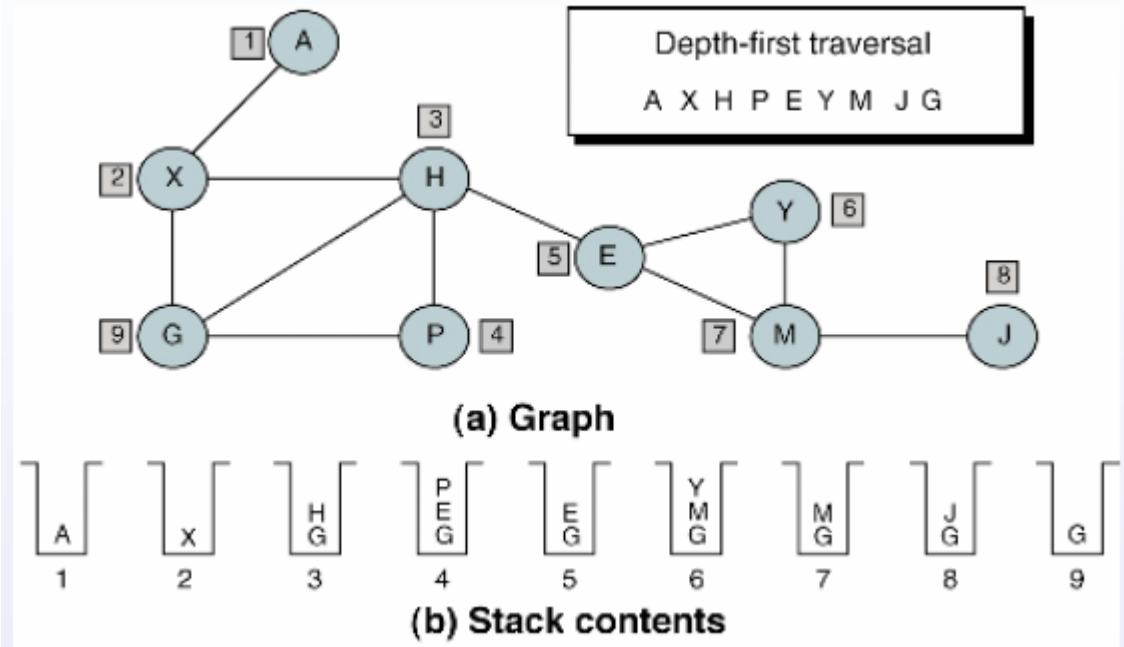
Depth First Search

- Pseudo Code

```
procedure DFS(G, v):
    label v as explored
    for all edges e in G.incidentEdges(v) do
        if edge e is unexplored then
            w ← G.opposite(v, e)
            if vertex w is unexplored then
                label e as a discovery edge
                recursively call DFS(G, w)
            else
                label e as a back edge
```

함수 재귀 호출의 스택 이용

Depth First Search



Node를 최대한 따라가다가 막히면 전에 들렀던 Node를 확인한다.

Breath First Search

- Breath First Search
  - Queue 기반
  - 시작 정점을 방문한 후 시작 정점에 인접한 모든 정점을 먼저 방문함
  - 더 이상 방문하지 않은 정점이 없을 때까지 나머지 정점에 대해서도 위의 과정 반복
- Time Complexity
  - O(|V| + |E|)
- Space Complexity
  - O(|V|)

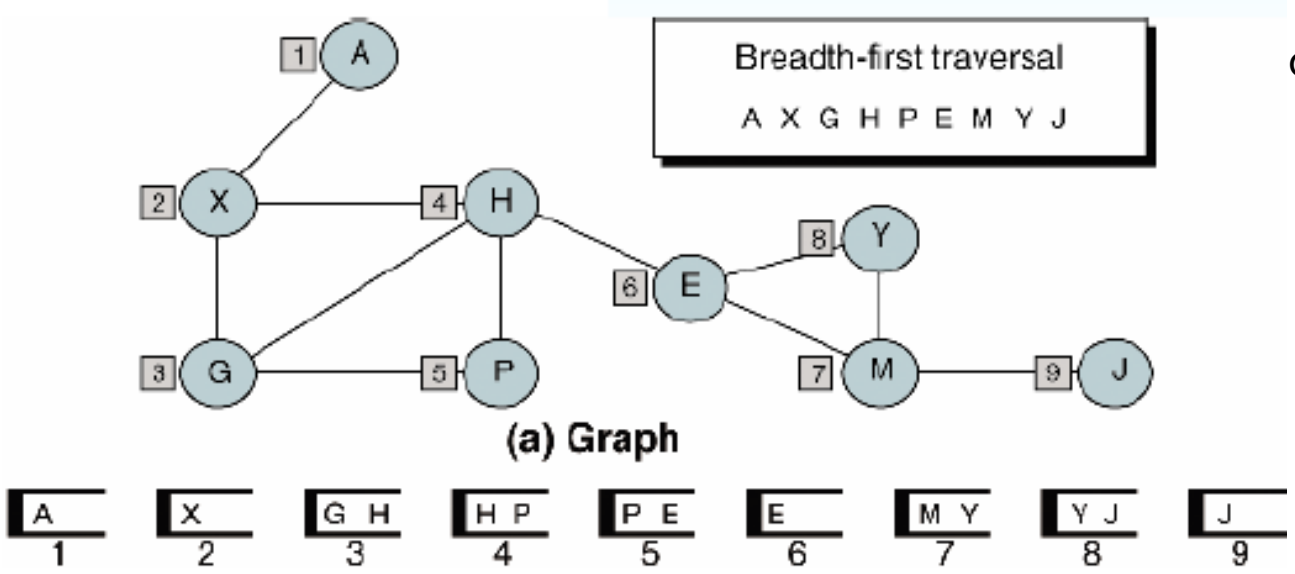
Breath First Search

- Pseudo Code

큐 이용

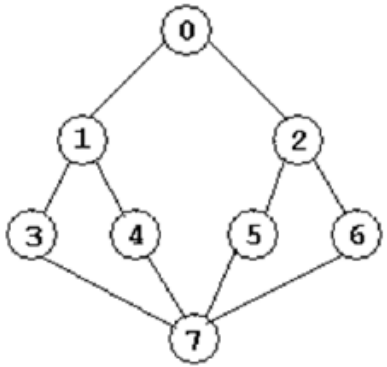
- **procedure** BFS(G, v):
  - create a queue Q
  - enqueue v onto Q
  - mark v
  - **while** Q is not empty:
    - $t \leftarrow$ Q.dequeue()
    - **if** t is what we are looking for:
      - return t
    - **for all** edges e in G.incidentEdges(t) **do**
      - $o \leftarrow$ G.opposite(t,e)
      - **if** o is not marked:
        - mark o
        - enqueue o onto Q

Breath First Search

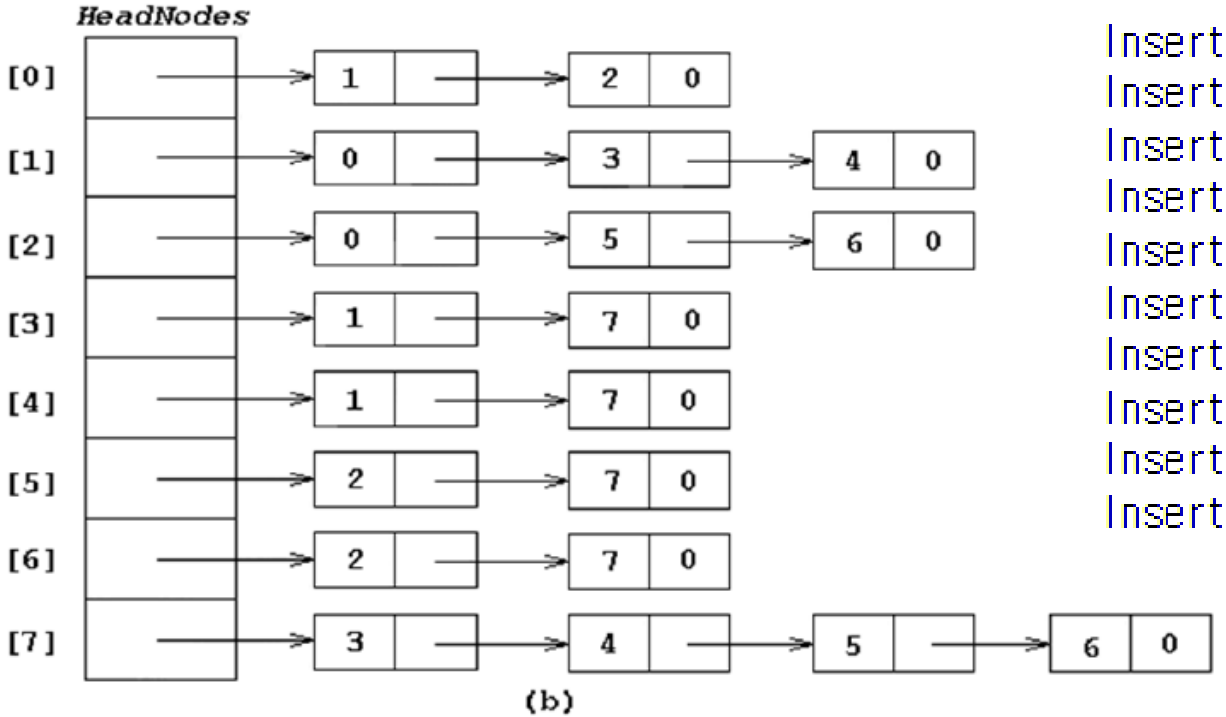


연결된 Link를 모두 확인한 후 다음 Node로 접근한다.

실습 모델 – Adjacency List



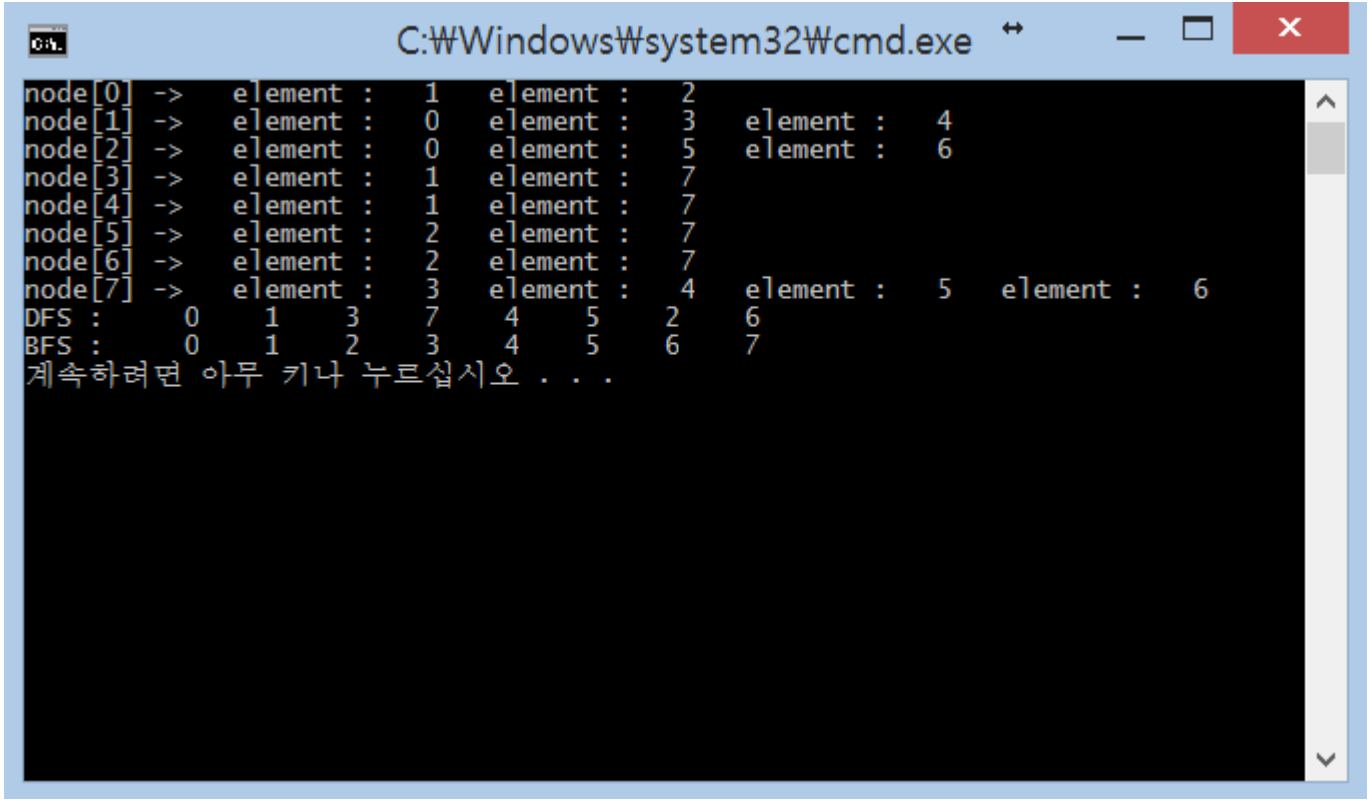
(a)



(b)

```
Insert(G,6,7);
Insert(G,5,7);
Insert(G,4,7);
Insert(G,3,7);
Insert(G,2,6);
Insert(G,1,4);
Insert(G,2,5);
Insert(G,1,3);
Insert(G,0,2);
Insert(G,0,1);
```

실습 결과 예


C:₩Windows₩system32₩cmd.exe
node[0] -> element : 1 element : 2
node[1] -> element : 0 element : 3 element : 4
node[2] -> element : 0 element : 5 element : 6
node[3] -> element : 1 element : 7
node[4] -> element : 1 element : 7
node[5] -> element : 2 element : 7
node[6] -> element : 2 element : 7
node[7] -> element : 3 element : 4 element : 5 element : 6
DFS : 0 1 3 7 4 5 2 6
BFS : 0 1 2 3 4 5 6 7
계속하려면 아무 키나 누르십시오 . . .

Data structure

```
typedef struct ListNode *position;
typedef position List;
typedef struct GraphTbl *GraphTable;

struct ListNode
{
    int Element;
    position Next;
};

struct GraphTbl
{
    int TableSize;
    List *TheLists;
};

int *visited;
#define true 1
#define false 0
```

ListNode는 각 key 값이 저장된 Node

GraphTable은 Node리스트를 가짐

다음 Node로 접근하기 위한 포인터 Next와 정보 Element가 있음

Visited는 행렬로 확인된 vertex인지 검사하기 위해 존재

CreateGraph (예시)

```
GraphTable createGraph(int size)
{
    GraphTable G;
    G = (struct GraphTbl*) malloc(sizeof(struct GraphTbl));
    G->TableSize = size;


    G->TheLists = (position*) malloc(sizeof(position)*size);

    for(int i=0; i<size ;i++)
        G->TheLists[i] = NULL;

    return G;
}
```

구조체 GraphTable 메모리 할당

Node 포인터를 갖는 배열을 생성

NULL 로 초기화

DFS

```
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

Visited[v]로 지나간 node 표시

링크를 계속 따라가면서 처음도착한 node에서 재귀함수로 구현

BFS

```
void bfs(int v)
{/* breadth first traveral of a graph, starting at v */
  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

Queue 초기화

Visited[v]로 지나간 node 표시

제출 및 알림

수업 중 확인 or 메일제출 (학번 써주세요)

메일 제출 :

주소 : (89kdsim@naver.com)

기한 : ~2016-06-08