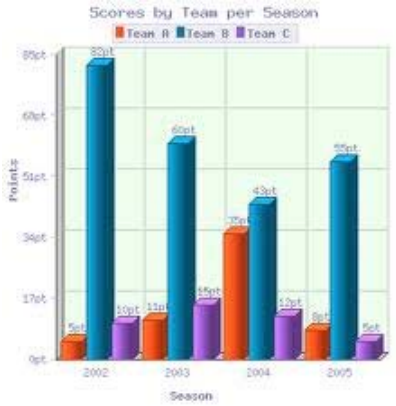
제 10 강의 . 그래프 개념과 그래프 탐색

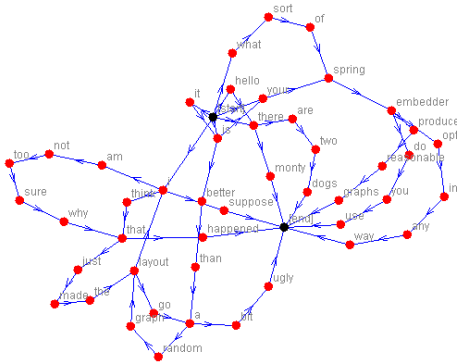
- 학습 목차

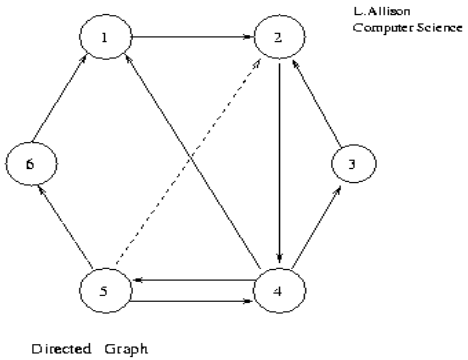
1. 그래프의 개념
2. 그래프의 표현
3. 그래프 탐색

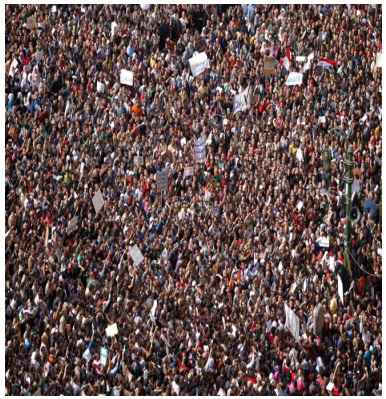
1. 그래프의 개념

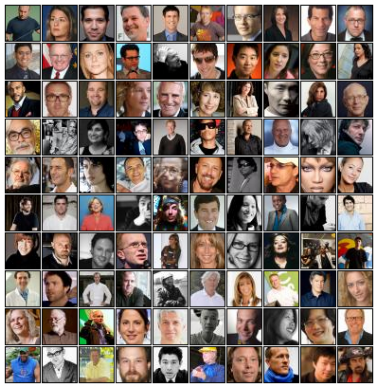
graph? chart?



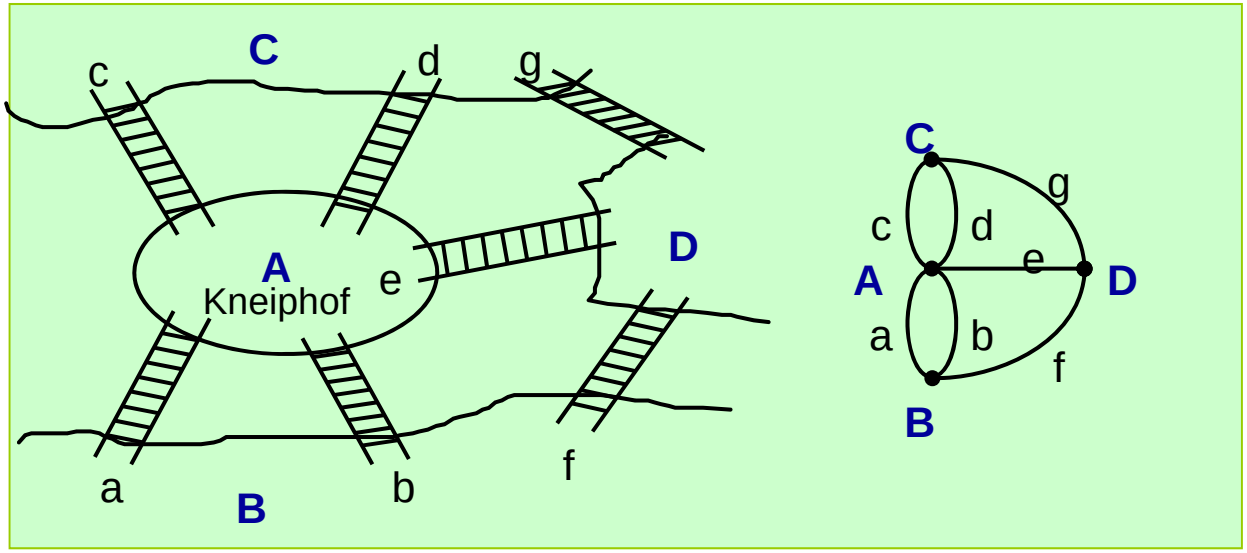








오래된 그래프 문제로 다음과 Köenigsberg 다리 문제가 있다. 이 문제는 다음과 같은 지형이 있을 때 임의의 한 곳(A,B,C,D)에서 출발하여 a부터 f까지 "모든 다리를 한번씩 건널 수 있는가?" 하는 문제이다. 자료구조의 그래프는 이러한 문제를 컴퓨터에 표현하고 알고리즘을 개발하는 분야이다.



Königsberg 다리 문제

1. 그래프의 개념

- **그래프의 수학적 정의**

그래프 : G = (V,E) 이고, V,E는 다음과 같다.

V(G) : 정점(set of vertices)

E(G) : 간선(set of edges), 정점을 연결하는 선, V X V의 부분집합

무방향 그래프(undirected graph) – 예를 들면 쌍방통행이 가능한 도로의 지도이다.

– 정점을 연결하는 선에 방향이 없다(undirected, unordered).

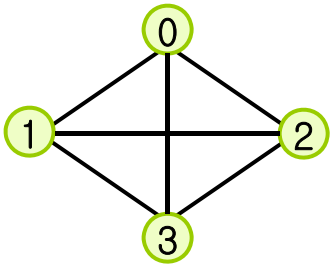
즉 $(v_i,v_j) = (v_j,v_i)$이다.

방향 그래프(directed graph) – 예를 들면 일방통행만 있는 도로의 지도이다.

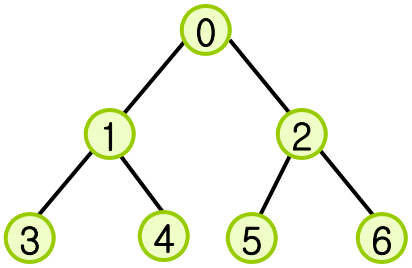
– 정점을 연결하는 선에 방향이 있다(directed, ordered).

즉 $\langle v_i,v_j\rangle \neq \langle v_j,v_i\rangle$

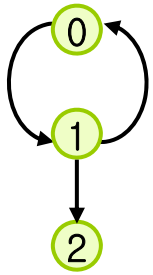
• **그래프의 예와 수학적인 표현** : 그래프를 표현하는 방법은 여러 가지이다. 아래 방법은 그림으로 그리는 그래프의 모습과 수학적인 기호로 표현하는 방법이다.



G_1



G_2



G_3

예 1) 그래프의 수학적인 표현

그래프 G_1

$V(G_1) = \{0,1,2,3\}$

$E(G_1) = \{(0,1),(0,2),(0,3),(1,2),(1,3),(2,3)\}$

그래프 G_2

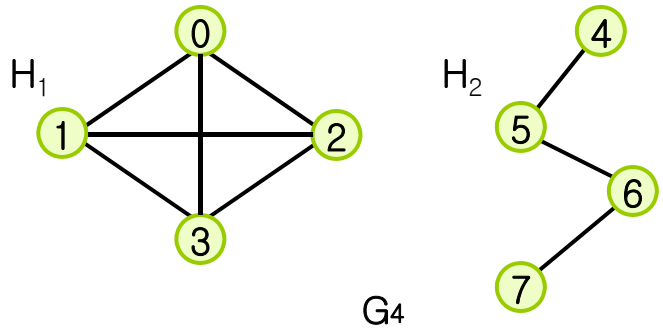
$V(G_2) = \{0,1,2,3,4,5,6\}$

$E(G_2) = \{(0,1),(0,2),(1,3),(1,4),(2,5),(2,6)\}$

그래프 G_3

$V(G_3) = \{0,1,2\}$

$E(G_3) = \{<0,1>,<1,0>,<1,2>\}$



예 2) 두개의 연결된 부분을 가진 그래프

그래프 G_4

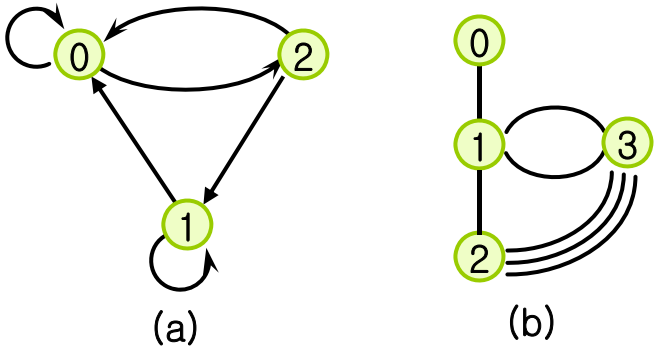
$V(G_4) = \{0,1,2,3,4,5,6,7\}$

$E(G_4) = \{(0,1),(0,2),(0,3),(1,2),(1,3),$
$(2,3),(4,5),(5,6),(6,7)\}$

• 그래프에서의 제한 사항

1) 자기 자신을 향하는 간선은 없다.(no self loop)
no edge from a vertex, i, back to itself, no (v_i,v_i) or <v_j,v_i>

2) 중복된 간선을 허용하지 않는다.(not multigraph)
no multiple occurrences of the same edge



그래프의 제한 사항의 예 a) self loops, b) multigraph

• 그래프에 관한 용어들

(1) 완전그래프(complete graph)

- 그래프에서 간선의 수가 최대인 그래프를 말한다.

정점이 3개인 그래프는 간선의 최대 갯수가 3개, 정점이 4개인 그래프는 간선의 최대 개수는 6개이다.

정점이 n개면? 계산을 해보자.

* 무방향그래프 경우 n 개의 정점이 있는 경우

간선의 수가 n(n-1)/2

* 방향그래프 경우 n 개의 정점이 있는 경우

간선의 수가 n(n-1)

(2) 인접(adjacent)

- 무방향 그래프에서 정점 a, b에 대하여 간선(a,b)가 있으면, 정점 a 는 정점 b에 인접(adjacent)하다고 한다.

(3) 부속(incident)

- 무방향 그래프에서 정점 a, b에 대하여 간선(a,b)가 있으면, 간선 (a,b)는 정점 a와 정점 b에 부속(incident)한다고 한다.

예) 무방향 그래프의 (v_0,v_1) : 무방향 간선

- 정점 v_0 과 v_1 은 인접(adjacent)하다.
- 간선 (v_0,v_1) 은 정점 v_0 와 v_1 에 부속(incident) 된다.

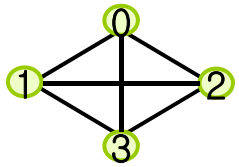
예) 방향 그래프 <v_0,v_1> : 방향 간선

- 정점 v_0 는 정점 v_1 에 인접(adjacent)하다.
- 정점 v_1 은 정점 v_0 로 부터 인접(adjacent)하다.
- 간선 <v_0,v_1> 은 정점 v_0 와 v_1 에 부속(incident)된다.

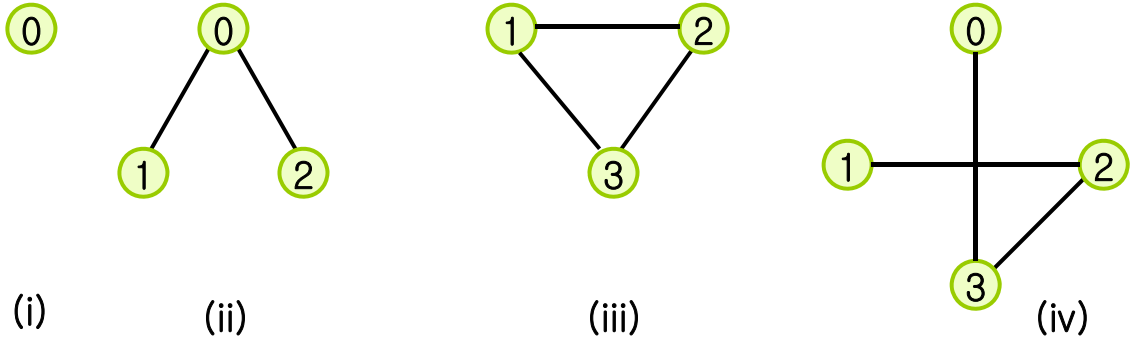
(4) 부분 그래프(subgraph) G′ of G :

- V(G′) ⊆ V(G) and
- E(G′) ⊆ E(G)

(a) 그래프 G_1의 부분 그래프의 예

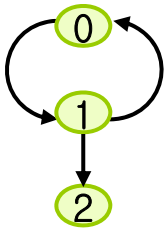


G_1

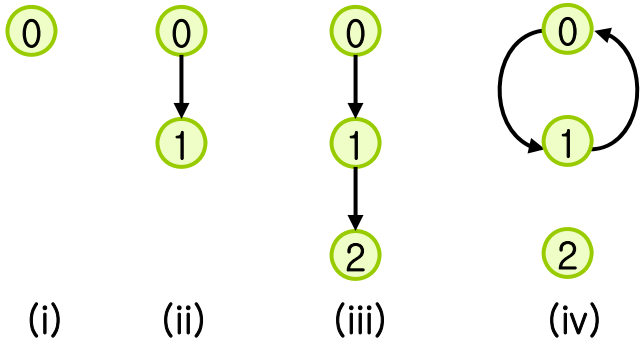


(i) (ii) (iii) (iv)

(b) 그래프 G_3의 부분 그래프들



G_3



(i) (ii) (iii) (iv)

•그래프에 관한 용어들

(5) 경로(path) : 정점 v_p 에서 정점 v_q으로 가는 경로

- 정점의 연속인 v_p, v_1, v_2, v_3, ···, v_n, v_q 가 다음과 같은 간선이 존재할 때 정점 v_p 에서 정점 v_q으로 가는 경로가 있다고 한다.

  (v_p,v_i), (v_1,v_2),···,(v_n,v_q)

- 무방향 그래프에서 다음의 간선들이 존재한다.

  $<v_p,v_1>$, $<v_1,v_2>$,···,$<v_n,v_q>$

(6) 경로의 길이 : 경로상에 있는 간선의 수

- (v_p,v_{i1}),(v_{i1},v_{i2}),···,(v_{in},v_q) 경로의 경우 n+1 이 된다.

(7) 단순 경로(simple path)

- 처음과 마지막을 제외하고 정점이 모두 다른 경로, 즉 경로상의 정점이 중복되지 않는 경로를 단순경로라고 한다.

(8) 사이클(cycle)

- 처음과 마지막 정점이 같은 단순 경로, 즉 단순경로 중 경로가 다시 원점에 도달하는 경우이며 사이클을 형성한다.
- 방향성 그래프에서는 방향성 사이클(directed cycle)이라 한다.

(9) 연결됨(connected)

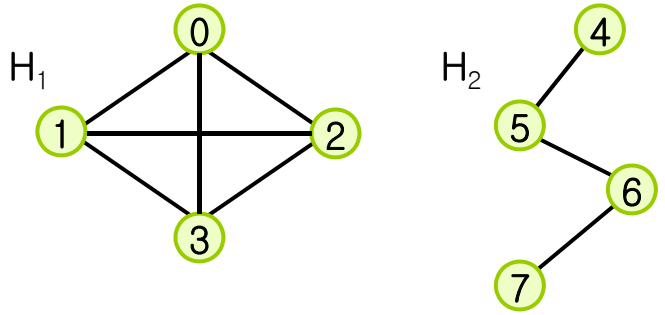
- 정점 v_0 와 정점 v_1 이 연결되있다는 것은 그래프 G에서 정점 v_0에서 정점 v_1 로 가는 경로가 있는 경우이다. 지도로 말하면 통행로가 있는 경우이다.

•그래프에 관한 용어들

(10) 연결된 부분(connected component)

- 부분 그래프에서 최대로 연결된 그래프, 즉 그래프에서 모든 정점이 연결되지 않을 수 있다. 이 경우 최대로 연결된 부분 그래프를 connected component라고 한다.

(maximal connected subgraph)



G4 – 두개의 연결된 부분을 가진 그래프

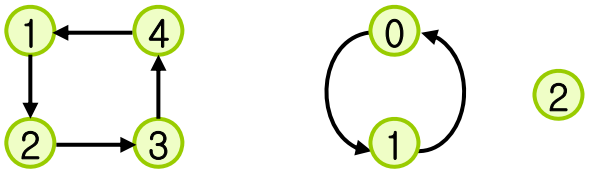
(11) 트리(tree)

- 트리를 그래프로 정의할 수 있다. 사이클이 없으며 연결된 그래프로 루트 노드가 1개 존재하는 그래프를 트리라고 한다. 트리는 그래프의 일종이다.

•그래프에 관한 용어들

(12) 강연결과 강연결 요소
(strongly connected, strongly connected component in a directed graph)

- 강연결 : 정점의 쌍 v_i, v_j in V(G)에 대하여 v_i to v_j 그리고 v_j to v_i 경로가 존재할 때 즉 정점의 상대방으로 가는 경로가 모두 존재할 때 강연결되있다고 한다.
- 강연결 요소(strongly connected component in a directed graph)
  강연결된 최대 부분 그래프, 정점의 모든 쌍 v_i, v_j in V(G)에 대하여 v_i to v_j 그리고 v_j to v_i 경로가 존재하는 연결된 부분 요소이다.(maximal subgraph that is strongly connected)



강연결된 그래프 G_3의 강연결 요소 (strongly connected components)

(13) 차수(degree of vertex)

- 정점에 부속된(incident) 간선의 수
- in-degree (of vertex v) : 정점에 들어오는 간선의 수
- out-degree (of vertex v) : 정점으로 부터 나가는 간선의 수

Q/A

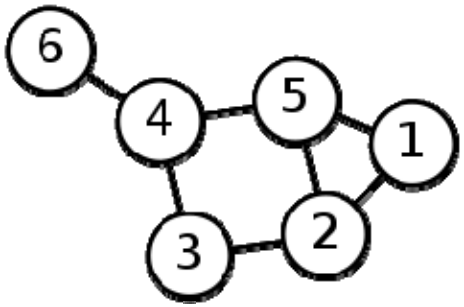
그래프에 대하여 답하여라

(1) Vertex와 edge의 수는 몇 개인가?

(2) 그래프를 수학적인 표기법으로 적어라.

(3) 5번 노드에 인접한 노드는?

(4) 6번에서 1번으로 가는 최단 경로의 길이는?



2. 그래프의 표현

(1) 인접 행렬(adjacency matrix)

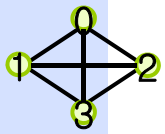
- 그래프 G = (V,E), |V| = n(≥1)일 때 그래프를 이차원 행렬에 다음과 같이 저장하는 방법이다.
  adj_mat[i][j] =

  1 : if (v_i, v_j)가 인접할 때(adjacent)

  0 : 인접하지 않을 경우

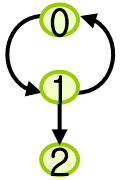
-필요한 기억장소의 크기 = 공간복잡도(space complexity) : S(n) = n^2

- 무방향 그래프에서는 행렬이 대각선을 중심으로 대칭(symmetric)이다.



G_1

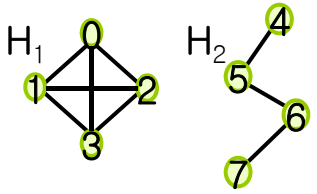
	0	1	2	3
0	0	1	1	1
1	1	0	1	1
2	1	1	0	1
3	1	1	1	0



G_3

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

G_4



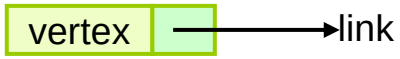
	0	1	2	3	4	5	6	7
0	0	1	1	1	0	0	0	0
1	1	0	1	1	0	0	0	0
2	1	1	0	1	0	0	0	0
3	1	1	1	0	0	0	0	0
4	0	0	0	0	0	1	0	0
5	0	0	0	0	1	0	1	0
6	0	0	0	0	0	1	0	1
7	0	0	0	0	0	0	1	0

인접 행렬로 표현된 그래프 G_1, G_3, and G_4

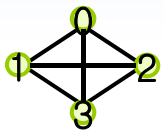
(2) 인접리스트(adjacency lists)

n 개의 연결리스트로 그래프를 표현한다. 그래프의 각 정점에 대하여 정점과 연결된 간선을 한 개의 연결리스트에 저장한다. 전체 연결리스트는 n개가 되며 이 연결리스트를 배열로 표현한다.

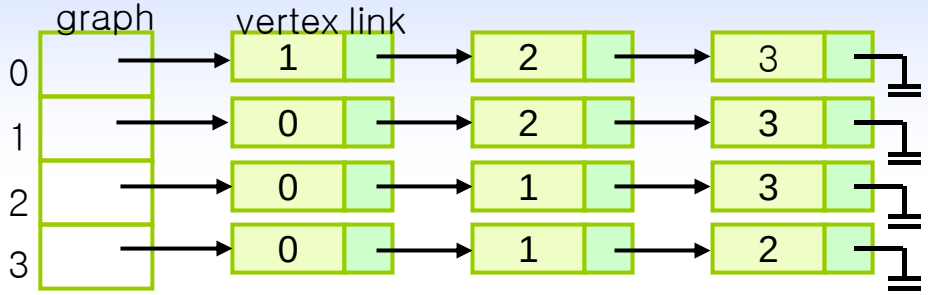
```
#define MAX_VERTICES 50
     struct list_node {
      int vertex;
      struct list_node * link;
     };
     typedef struct list_node node;
     typedef node * node_ptr;
node_ptr graph[MAX_VERTEX];
```



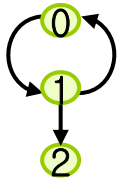
노드의 자료구조



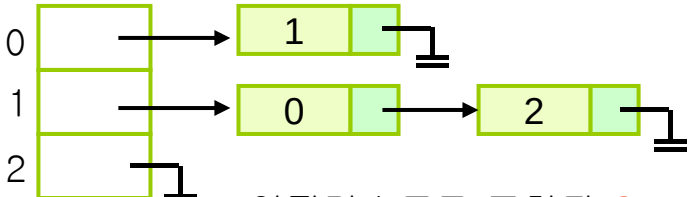
G_1



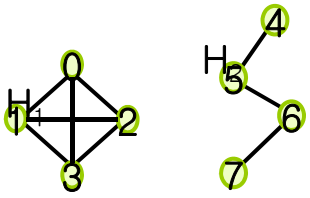
인접리스트로 표현된 G_1

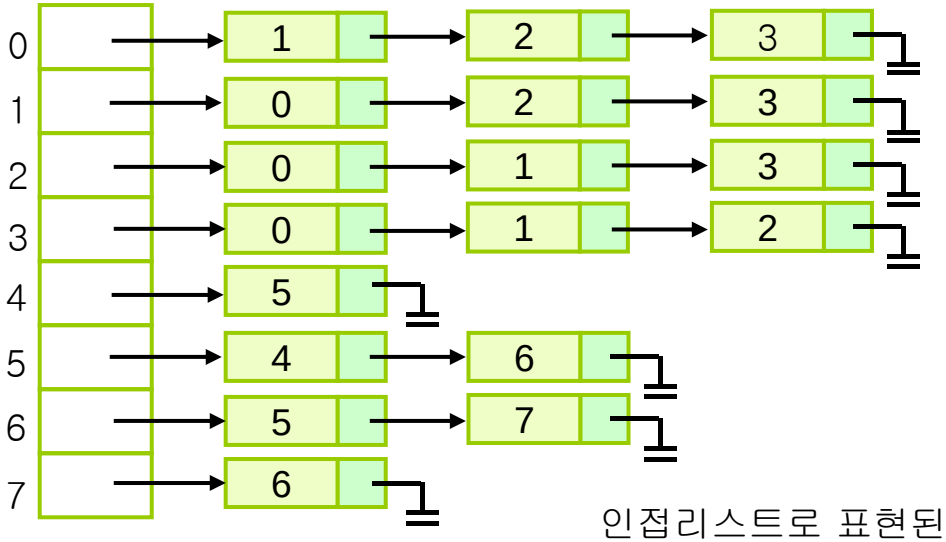


G_3



인접리스트로 표현된 G_3

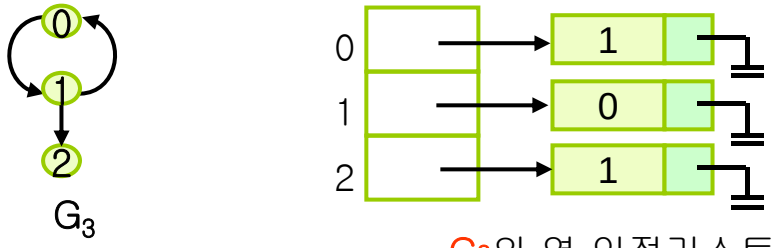




인접리스트로 표현된 G_4

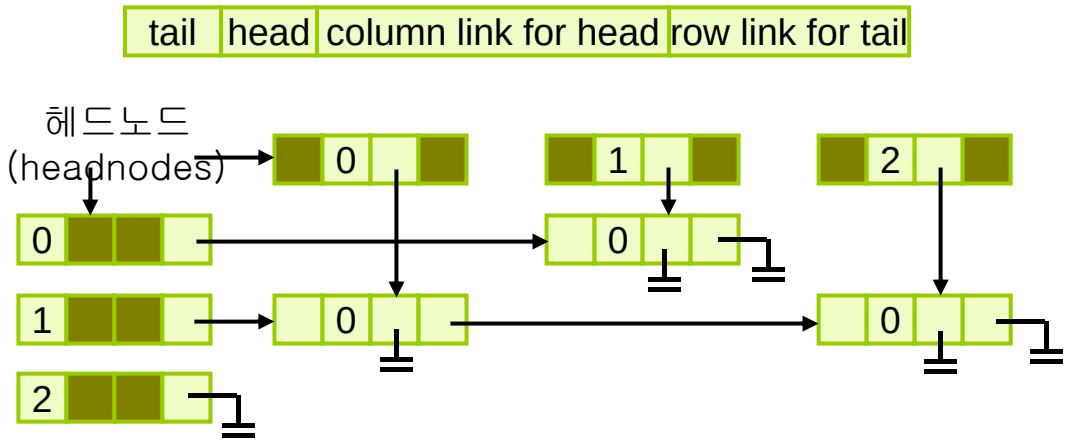
(2) 인접리스트(adjacency lists)

참고 1 : 역(inverse) 인접리스트



G_3의 역 인접리스트

참고 2 : 2차원 직교 표현 – 연결리스트를 이차원으로 표현한다.



그래프 G_3의 직교 표현

참고 3 : 가중치 간선(weighted edges)

- 그래프의 간선에 가중치(weights)를 부여할 수 있다. 가중치의 의미는 다음이다.

  1) 정점에서 정점으로 가는 거리 혹은

  2) 정점에서 정점까지의 도달하는 비용

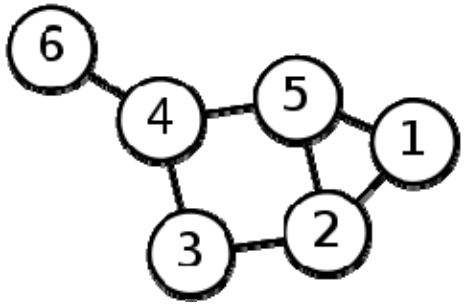
- 인접 행렬이나 인접리스트에 가중치를 부여하여 표현한다.

Q/A

그래프에 대하여 답하여라

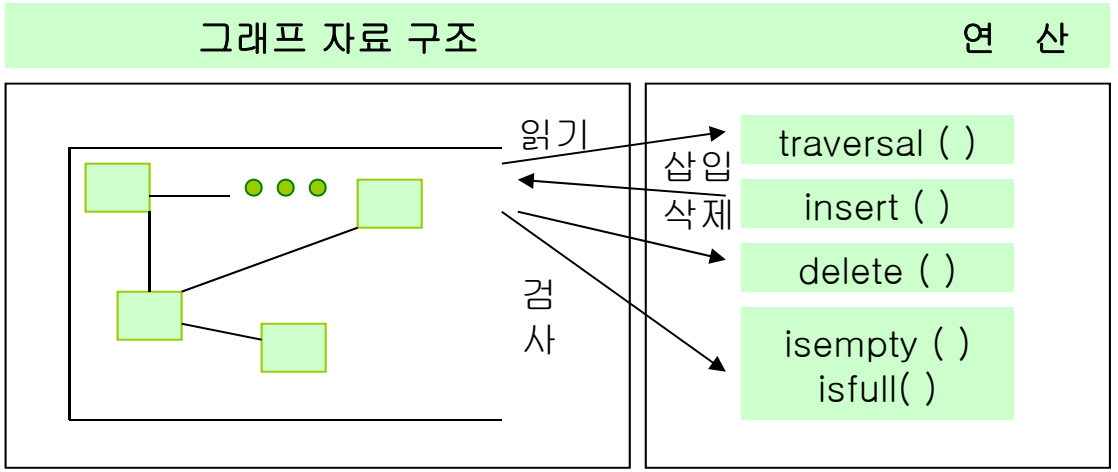
(1) 그래프를 인접행렬로 나타내어라

(2) 그래프를 인접리스트로 나타내어라



3. 그래프 탐색

❖ 그래프 자료구조와 연산 모델 : 그래프를 표현하고 그래프에 관한 연산을 구현한다.



그래프 자료구조 = 그래프 자료선언 + 그래프 연산

그래프 자료구조는 자료의 선언과 자료에 대한 연산으로 구성된다.
그래프 자료의 선언과 저장은 인접 행렬이나 인접리스트로 표현한다.
그래프 연산은 자료의 탐색(traversal)과 갱신에 관한 연산으로 이루어 진다.
탐색은 원하는 노드를 1개 혹은 일부를 찾는 작업이다.
갱신은 정점과 간선의 삽입(insert), 삭제(delete), 수정을 말한다.

•그래프 탐색(graph traversals)

- 그래프의 각 정점을 방문하는 것을 탐색(traversal)이라고 한다.
- 그래프에 관한 연산 중 가장 중요한 것이다.
- 탐색에서 노드의 방문 순서에 따라 다음과 같은 방법이 있다.

• 깊이우선탐색 – DFS (Depth First Search) – 갈수있는데 까지 가보는 방문 방법

–트리의 전위탐색 방법을 그래프에 적용한 것이다.

(preorder tree traversal)

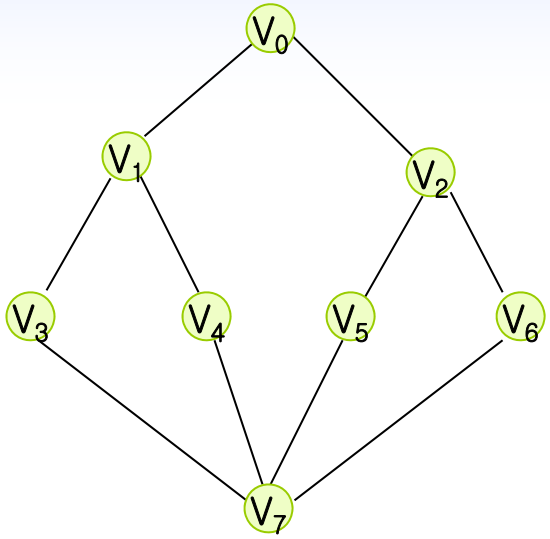
• 너비우선탐색 – BFS (Breath First Search) – 거리가 가까운 곳부터 가보는 방문 방법

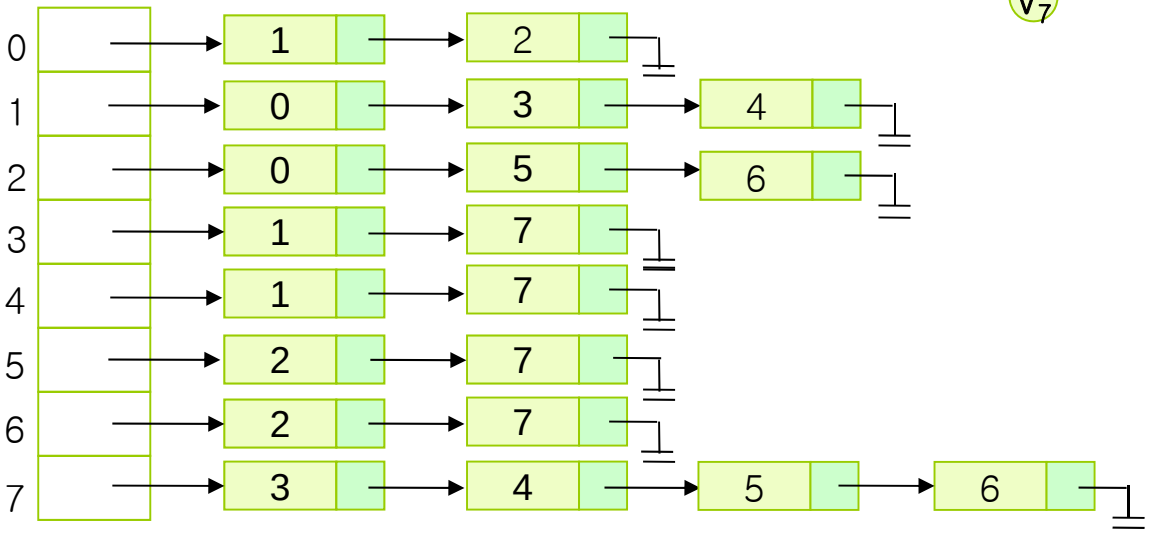
– 트리의 레벨우선탐색을 그래프에 적용한 것이다.

(level order tree traversal)

3. 그래프 탐색

(a) 그래프 G





(b)

그래프 G와 인접리스트(adjacency lists)

(1) 그래프 탐색 - 깊이우선탐색(depth first search)

방법 : 자동차로 여행할 경우 방문지가 있으면 무조건 방문하는 방법이다.
즉 후진하지 않고 가는 방법이며 후진하는 경우는 길이 막히거나, 이미 방문했던 곳일 경우이다. 후진하여 방문할 곳은 마지막에 갈수 있었던 곳 중 가지 않았던 길이다. 이 방법을 깊이 우선 탐색이라고 하며 탐색 중 방문 가능한 곳은 스택 자료를 이용하여 저장해 둔다.

1단계 : 하나의 노드를 택한다.

2단계 : 노드를 방문하여 필요한 작업을 한 다음 연결된 다음 노드를 찾는다. 현재 방문 노드는 스택에 저장한다. 2단계를 반복하면서 방문을 계속한다. 막히면 3단계로 간다.

3단계 : 더 이상 방문할 노드가 없으면 스택에서 노드를 빼내 다음 방문 노드를 찾아 2)번 과정을 다시 반복한다.

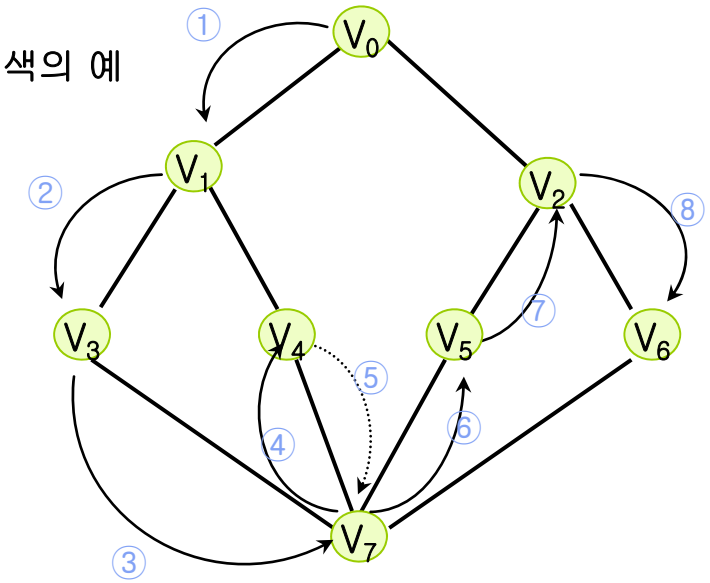
특징 :

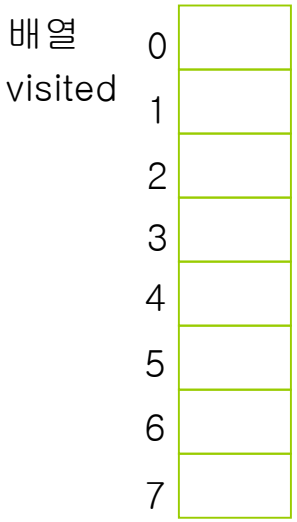
- 스택을 이용한다.
- visited[MAX_VERTICES] 배열을 사용하여 방문했던 것을 표시한다.
  초기 값은 FALSE 이고 방문하면 TRUE

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

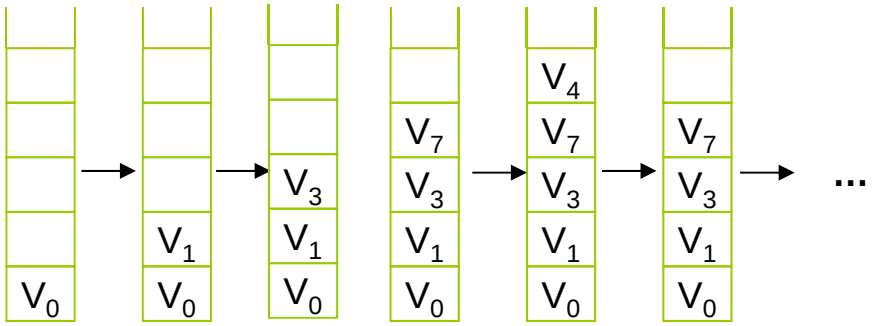
(1) 그래프 탐색 – 깊이우선탐색(depth first search)

깊이우선탐색의 예





- ①②③④⑥⑦⑧ : 방문
- ⑤ : 되돌아옴
- 방문할 곳이 여러 곳일 때 노드번호가 작은 쪽 방문



스택의 변화

깊이우선탐색 알고리즘(depth first search)

```
/* 깊이우선탐색 알고리즘(depth first search) */
/* 정점 v에서 시작하여 그래프의 깊이 우선 방문 */
void dfs(int v) {
        node_ptr w;
        visited[v] = TRUE; /* 1 방문지 표시 */
        printf("%5d", v);
        for(w = graph[v]; w; w = w->link) /* 2 연결리스트 탐색 */
                if(!visited[w->vertex])
                        dfs(w->vertex);
}
```

* 깊이우선탐색 알고리즘(depth first search)의 시간복잡도

  -> 시간복잡도(time complexity) : O(e), e는 간선의 수

(2) 그래프 탐색 - 너비우선탐색(breadth first search)

방법 : 자동차로 여행할 경우 가까운 곳 부터 방문하는 방법이다. 즉 어느 지점에서 가까운 곳을 방문하고 다음 방문 가능 지점은 모두 저장해둔다. 다음 방문지는 항상 저장해둔 곳에서 순서대로 꺼낸다. 예를 들어 종로 1가에서 시작하여 종로의 버스정류장을 모두 방문해야 한다고 하면 종로 1가의 한 정류장 거리를 다 방문하고 끝나면 두 정류장 거리에 있는 정류장을 모두 방문하고 이러한 순서로 진행한다. 두 정류장 거리에 있는 정류장들은 첫번째 정류장을 방문할 경우 기억을 해둔다. 먼저 기억된 정류장이 다음 번에 먼저 방문을 하게 된다. 즉 큐 자료구조를 이용하여 다음 방문지를 선택한다.

1단계 : 하나의 노드를 택한다.

2단계 : 노드를 방문하여 필요한 작업을 한 다음 연결된 다음 노드를 찾는다.
노드들을 큐에 저장한다. 더 이상 방문할 곳이 없으면 3단계로 간다.

3단계 : 큐의 맨 앞의 노드를 빼내 2단계를 반복한다.

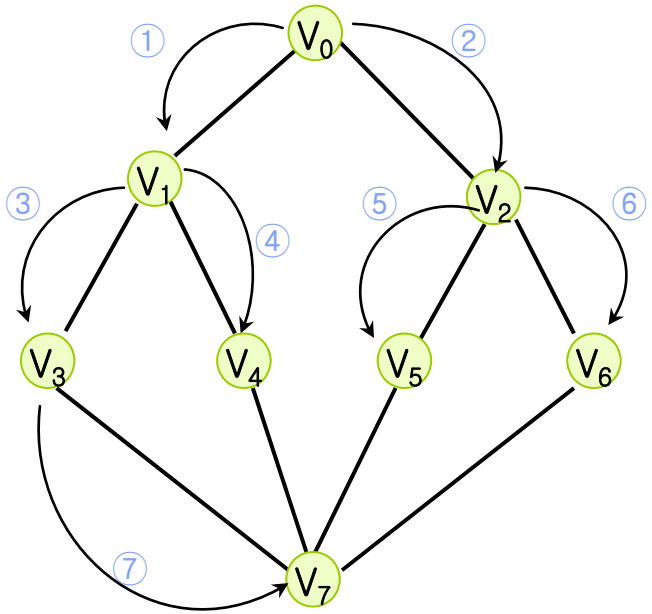
특징 :

- 큐를 이용한다.
- visited[MAX_VERTICES] 배열을 사용하여 방문했던 것을 표시한다.
  초기 값은 FALSE 이고 방문하면 TRUE

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

너비우선탐색(breadth first search) 예제

너비우선탐색의 예



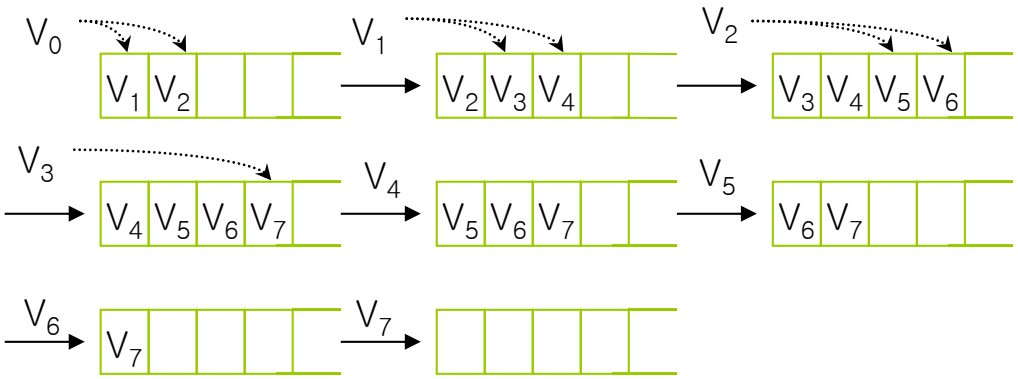
배열 visited

0	
1	
2	
3	
4	
5	
6	
7	

①②③④⑤⑥⑦ : 방문

방문할 곳이 여러 곳일 때 노드번호가 작은 쪽 방문

너비 우선 탐색의 예 – 큐의 모양



너비 우선 탐색 알고리즘 (breadth first search) */

```
/* 너비 우선 탐색 알고리즘(breadth first search) */
/* 정점 v에서 시작하여 그래프의 너비 우선 방문 */
void bfs(int v) {
        node_ptr w; queue_ptr front, rear;
        front = rear = NULL;/* initialize queue */
        printf("%5d", v);
        visited[v] = TRUE; /* 1 방문지 기록 */
        insert(&front, &rear, v);
        while(front) /* 2 방문지 있는 동안 반복 */
        {  v = delete(&front);
                for(w = graph[v]; w; w = w->link;
                if(!visited[w->vertex]) {
                        printf("%5d", w->vertex);
                        insert(&front, &rear, w->vertex);
                        visited[w->vertex] = TRUE;
                }
        }
}
```

(3) 연결 요소의 계산 – bfs(), dfs()를 이용 하여 그래프의 어느 한 노드에서 연결되는 모든 노드를 구하는 연결 요소(connected components) 을 구한다.

- 알고리즘 *dfs*(0) or *bfs*(0)를 호출하여 구한다.
- dfs()나 bfs()는 경로가 있는 노드들만을 탐색한다.

연결 요소의 계산

```
void connected(void) {
/* 그래프의 연결 요소를 출력한다 */
    int i;
    for(i = 0; i < n; i++) {
        if(!visited[i]) {
        dfs(i);
        printf("₩n");}
    }
}
```

그래프 G_4 에 알고리즘을 적용하면 결과는 2줄이 된다.
즉 2개의 연결 요소를 찾는다.

(결과) 0 1 2 3
4 5 6 7

학습내용정리

⦿ 그래프는 지도상의 최단경로 찾기 등의 문제를 해결하는데 사용이 된다. 그래프 자료구조는 그래프를 컴퓨터에 표현하고 그래프에 관한 알고리즘을 개발하는 기초이다.

⦿ 그래프는 정점(node)과 간선(edge)의 집합으로 구성이 된다. 간선이 방향을 갖는지 여부에 따라 무방향 그래프와 방향 그래프로 구분한다. 그래프에 관한 용어로는 경로, 사이클, 연결요소 등이 있다.

⦿ 그래프를 컴퓨터에 저장하는 방법은 인접 행렬과 인접리스트가 있다. 인접행렬은 행렬로 표현하는 방법이며 인접리스트는 연결리스트를 이용한 방법이다. 인접 행렬은 표현하기 쉽지만 기억장소의 낭비가 많기 때문에 잘 사용되지 않는다. 연결리스트는 복잡하기는 하지만 큰 그래프를 표현할 수 있는 방법이다.

⦿ 그래프에 관한 연산은 트리와 마찬가지로 탐색이 가장 중요하며 탐색 방법은 깊이우선탐색과 너비우선탐색의 두 가지 방법이 있다. 두 방법 모두 순환 알고리즘을 이용하여 작성하며 이미 배운 스택과 큐 자료구조를 이용하게 된다.